TPLReport™

Version 8.0

QQQ Software, Inc.

302 N. Irving Street
Arlington, VA 22201 USA

tel 703-528-1288
fax 703-528-1289

http://www.qqqsoftware.com
support@qqqsoftware.com

TPL Report User Manual Version 8.0

This manual may be used with licensed copies of TPL Report and demonstration versions of TPL Report.

QQQ Software, Inc.
302 N. Irving Street
Arlington, VA 22201 USA
Tel: 703-528-1288
Fax: 703-528-1289
email: support@qqqsoftware.com
web: http://www.qqqsoftware.com

August, 2014

Preface

TPL REPORT is a report writer that shares many of the features of TPL TABLES. It can be used with the same data and codebooks (data descriptions). Language conventions are the same and the mode of operation is similar.

Information about data and codebooks that is particularly relevant to TPL REPORT is contained in a single chapter called "Data and Codebooks". Complete details on these subjects can be found in the TPL TABLES manual in the chapters called "Data" and "Codebooks".

TPL REPORT Version 8.0 is available for PCs running Windows XP, Vista, Windows7, Windows8 and for a variety of UNIX and Linux platforms.

Unless specifically noted, the information in this manual applies to both Windows and UNIX versions of TPL REPORT.

If you are viewing this document with Adobe's Acrobat Reader, you can click on entries in the Table of Contents or Index to be transferred to the pages in the text.

All of the report examples in this user manual were produced using TPL REPORT's PostScript output features. The reports were then inserted into the text using a desktop publishing system.

We hope that you will enjoy using TPL REPORT and will write or call if you have questions, comments or suggestions. Your comments are important to us and will guide our selection of features to include in future versions of TPL REPORT.

Contents (Summary)

Contents

Totals 62

Select 71

Compute 82

Recode 92

Char 103

Hierarchies 105

Repeating Groups 123

Labels 129

Masks 148

PostScript 155

Color and Grey 162

Exports 171

TPL-SQL 173

Run Instructions (Windows) 312

Scripts (Windows) 322

Installation (UNIX) 335

Run Instructions (UNIX/Linux) 340

CHAPTER 1

Introduction

WHAT IS TPL REPORT?

TPL REPORT is a report writer that lets you examine your data in detail. You can request a record by record listing of any or all of the variables. The data will be displayed in rows that can match the order of the data file or be sorted according to the values of one or more variables. The values for each variable are displayed in columns in the order of your choice, and you can request subtotals and totals.

TPL REPORT can report data from files of many different formats, including hierarchical files. It can process an unlimited amount of data and produce reports that range in size from a few lines to hundreds or thousands of pages. Subsets of the data can be selected and new variables can be computed from existing data. Values can be recoded in a variety of ways.

TPL REPORT will automatically format your reports if you wish. In addition, you can choose from the many optional format commands. For example, reports can be altered by deleting columns, changing labels and titles, adding horizontal and vertical lines, and changing the format of the values. One format command strips the report of everything but the data, thus producing an ASCII data file output that can be used as input to other software packages.

TPL REPORT can format your reports using PostScript® fonts. The PostScript reports can be printed on any laser printer or typesetting machine that processes PostScript. This feature gives you a choice of type style and size, including proportional fonts. The reports can be printed directly or incorporated in documents that have been created with desk-top publishing software.

If you use TPL TABLES, you will find it very easy to learn and use TPL REPORT. *You can use the same data and codebooks* for both, and the style and types of statements are very similar. The fundamental difference between the two is that TPL TABLES tabulates data, while TPL REPORT can give you a record by record

listing of the individual data items. This means, for example, that you can tabulate data, then look at the individual items that contribute to selected table cells.

How Does TPL REPORT Work?

The ingredients needed to create reports are: a data file, a codebook that describes the data file, and a report request that describes the reports. An optional ingredient is a format request that makes changes to the automatic report formats.

The Data File

TPL REPORT can work with data files from a variety of sources. For example, the data can be exported from a data base or spreadsheet, downloaded from a mainframe, or prepared using an editor or data entry system. TPL REPORT does not prepare the data, import it into a format of its own, or change the data in any way. It simply reads it and extracts the information needed to produce the reports you want.

The Codebook

The first step in creating reports from a particular data file is to prepare a codebook that describes your data. If you have already written a codebook for TPL TABLES, you can skip this step. The codebook you already have will work with no changes.

The codebook contains information such as the names of data fields, where they are located within a record and how many character positions (bytes) each occupies within a record. Since TPL REPORT does not require that your data be in a particular format, it needs this information in order to find the data values that you wish to use in your reports.

The codebook is a text file that can be prepared with an editor. In the Windows version of TPL REPORT, you also have the option of preparing the codebook interactively. For the UNIX version, the *tpl conditions* program can assist you in preparing the codebook.

After you have prepared the codebook, TPL REPORT will process it and convert it to a form that it can use to work with the data. When this process is complete, you can use the codebook over and over to create any number of reports from the data file.

The Report Request

The second step in creating reports is to prepare a report request. The report request is a file that you prepare using your editor. It contains TPL statements that describe the reports you want. You can reference any of the variables in your codebook by name. In addition, you can select subsets of the data file, compute new variables and recode existing variables.

The most important statement in a report request is the one that describes the structure and content of a report. You can request one or many reports in the same report request.

Once you have prepared the codebook and the report request, TPL REPORT can read your data to produce the reports you have requested. It will automatically format the rows and columns of the reports in the order directed by the report statements, using names and labels from the codebook and report request. If your report is too wide to fit across the page, TPL REPORT will break it into pages that can be placed side by side for easy review.

The Format Request

An optional third step is the preparation of a format request. Like the report request, the format request is a file that you prepare with your editor. It contains FORMAT statements that you use to make changes to the automatic report format.

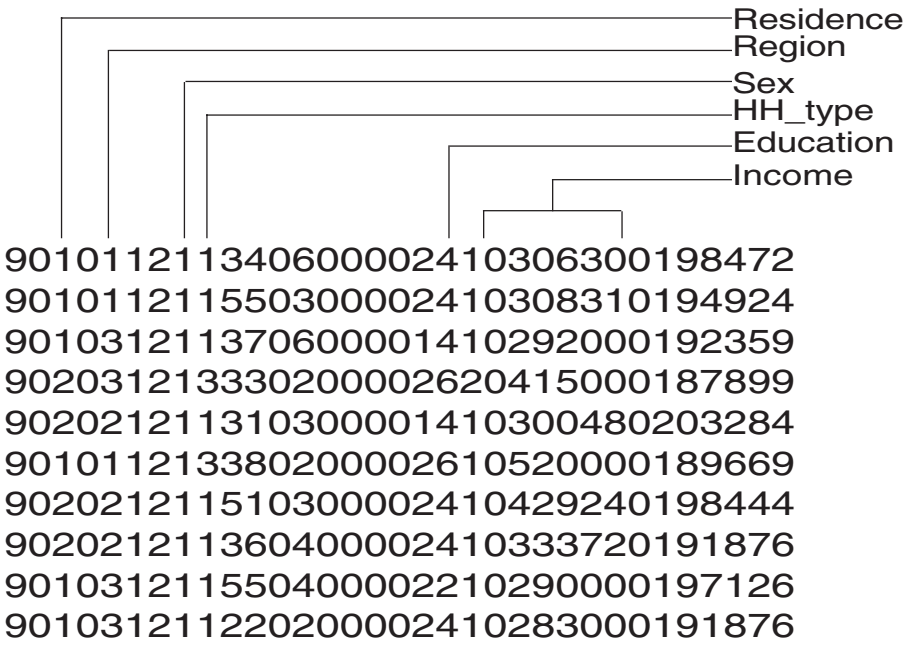
The automatic formats provided by TPL REPORT are usually acceptable for a quick look at the data. However, you may wish to change such things as column widths, labels or alignments. These report characteristics can easily be changed with FORMAT statements.

An Example

Following is an example that illustrates how a data file, a codebook and a report request work together in TPL REPORT.

Data

First is a small sample of ten records from a data file that contains information about households. Each record in the data file represents one household.



Codebook

Next is the codebook that describes the data items that we plan to use with TPL REPORT. Each data item is described in the order of its occurrence on the data record. FILLER entries account for the parts of the record that we do not plan to use.

```
BEGIN  HH  CODEBOOK

HOUSEHOLDS  'Households'  MASK  99,999  RECORD

FILLER 2

RESIDENCE  'Type of Residence'  CONTROL  1
  (
        'Inside metropolitan areas'       = 1
        'Outside metropolitan areas'      = 2
  )
FILLER  1
REGION  CONTROL  1
 (
        'Northeast' = 1
        'West' = 2
        'South' = 3
  )
FILLER 2
```

```
SEX 'Sex of Householder' CONTROL 1
  (
        'Male' = 1
        'Female' = 2
  )
HH_TYPE  'Type of Household'  CONTROL  1
  (
        'Married couple'          = 1
        'Other family'            = 2
        'Nonfamily household'     = 3
  )
FILLER  9

EDUCATION   'Education of Householder'  CONTROL  1
  (
        '8 years or less'                 = 1
        'Some High School'      = 2
        'High School Graduate'  = 3
        'Some College'          = 4
        'College Graduate'      = 5
        'Post Graduate'         = 6
  )
FILLER  1

INCOME   'Income'  MASK 999,999  OBS  6

FILLER  7

END  HH  CODEBOOK
```

Report Request

The following report request begins with a USE statement that references the name of the codebook to be used with the data. The SELECT statement specifies that only records with INCOME values less than 30,000 be used in the report. There is one REPORT statement. It lists the variables to be displayed in the report. In this statement, the data values for REGION and INCOME are to be displayed exactly as they appear in the data file. The other variable, HH_EDUC is a recode of EDUCATION that replaces all values with their labels in the report. The report is to be sorted by EDUCATION within REGION.

```
USE  HH  CODEBOOK;

SELECT IF INCOME < 30000;

RECODE HH_EDUC 'Education'  ON  EDUCATION;
LABEL  IF  ALL;
```

```
REPORT ONE 'Report Q1.  Household data for households with income '
'below $30,000, sorted by education of householder within each region.':
REGION THEN HH_EDUC THEN INCOME;

SORT ONE  ON  REGION  EDUCATION;
```

The Report Output

TPL REPORT reads the report request, the data file and the codebook. It uses the codebook to find the required items in the data file and then selects the appropriate records. It automatically formats a report that lists the values requested in the REPORT statement, sorting them as specified. The labels used in the report are taken from the codebook and report request.

Following are the first fifteen rows of the report.

Note that row numbers are automatically inserted at the beginning of the report We could delete these from the report if we wished.

Report Q1. Household data for households with income below $30,000, sorted by education of householder within each region.

Row	Region	Education	Income
1	1	Some High School	6,700
2	1	Some High School	29,000
3	1	High School Graduate	28,300
4	1	High School Graduate	8,100
5	1	High School Graduate	9,976
6	1	High School Graduate	29,200
7	1	High School Graduate	22,000
8	1	Some College	7,100
9	1	Post Graduate	12,551
10	2	Some High School	18,200
11	2	High School Graduate	18,200
12	2	High School Graduate	2,676
13	2	High School Graduate	11,378
14	2	High School Graduate	15,877
15	2	High School Graduate	7,354

Format Request

In the above report, the alignments and column widths are set according to the automatic defaults. If we wish, we can add a format request to make our own choices for the report format. In the following format request, we add some horizontal lines, choose different alignments for different columns, and increase the columns width so that the Education values do not need to wrap.

Note that the column containing the row numbers is Column 0.

```
RETAIN CROSS RULES;
ALIGN COLUMNS LEFT;
FOR COLUMN 3:  ALIGN COLUMN CENTER;
FOR COLUMN 2:  COLUMN WDTH = 20;
```

Report Q1. Household data for households with income below $30,000, sorted by education of householder within each region.

Row	Region	Education	Income
1	1	Some High School	6,700
2	1	Some High School	29,000
3	1	High School Graduate	28,300
4	1	High School Graduate	8,100
5	1	High School Graduate	9,976
6	1	High School Graduate	29,200
7	1	High School Graduate	22,000
8	1	Some College	7,100
9	1	Post Graduate	12,551
10	2	Some High School	18,200
11	2	High School Graduate	18,200
12	2	High School Graduate	2,676
13	2	High School Graduate	11,378
14	2	High School Graduate	15,877
15	2	High School Graduate	7,354

CHAPTER 2

Overview

AN OVERVIEW OF TPL REPORT FEATURES

This chapter provides a brief introduction to the basic TPL REPORT features. The features are described in the approximate order of the chapters and appendixes of the User Manual.

Data Files

TPL REPORT reads data from sequential data files. File structures can be either "flat", containing only one type of record, or hierarchical. Hierarchical files contain a variable number of related records of increasing detail. Data can be stored as ASCII characters or as binary or floating point numbers. Only one data file format, described by a single codebook, can be processed at one time by TPL REPORT. Multiple data files with the same file format can be processed in one job. TPL REPORT can also read CSV and other types of delimited data files. With the TPL-SQL option, TPL REPORT can read data from a variety of databases.

Describing the Data

The input data file is described to TPL REPORT by means of a codebook. The codebook describes the file structure, naming each record that makes up a processing unit. Each data item in the record is called a variable. A variable has a name and a type that indicates the types of values that can occur for that variable.

The codebook is created as a separate step before reports can be produced. Once the codebook is created, it can be referenced any number of times.

Defining the Structure and Content of a Report

The REPORT statement lets you choose the variables to be displayed as columns in the report and the order in which the columns should appear. For very quick results, you can request a report of ALL variables or a report of ALL EXCEPT certain variables.

Sorting the Report

SORT statements can be used to sort reports. Reports can be sorted on one or more variables in either ascending or descending order.

Totals

The GRAND TOTAL statement lets you request a total row at the end of a report. The SUBTOTAL statement lets you request subtotal rows at any level of a *sorted* report. In either type of statement, you can choose the columns that will have totals. The same report can contain both subtotals and grand totals.

Selecting Subsets of the Data

A SELECT statement can be used to report on only a subset of the data file. Data can be selected based on data values, or certain sections or percentages of the data can be selected. The SELECT statement can contain combinations of logical and arithmetic tests. Multiple tests on several variables can be strung together with AND's and OR's.

Computing New Values

COMPUTE statements can be used to create new variables by combining variables from the data file with arithmetic operations. Alternate computations can also be requested, depending on whether specified conditions are met. Arithmetic operations allowed in computations include addition, subtraction, multiplication, division, and exponentiation, plus the absolute value and square root functions.

Recoding Data

TPL REPORT provides a very powerful RECODE statement for replacing values with new values or combinations of labels and values. For example, income amounts can be classified by ranges and code values assigned to the categories. Or value labels can be displayed in reports in place of values to increase readability.

Labels

Descriptive print labels can be assigned as report titles, variable labels and value labels. Labels can include spaces, upper and lower case letters, and special characters. Break points can be chosen for multiline labels, and alignment can be specified. In PostScript mode, you can vary the type styles within labels.

Masks

Masks can be used to control the format of values printed in report cells. With a mask, you can format data to show decimal places, include special characters such as dollar signs and percent symbols, and specify the alignment of data within a column. In PostScript mode, you can choose the type style for report cells.

Report Formatting

Reports can be formatted automatically, but, in addition, many details of report format can be adjusted with FORMAT statements. Column widths can be altered, reports can be split into sections on the same page, separate reports can be combined onto the same page, and extensive relabeling can be done. Reports can be prepared for publication with PostScript, allowing type size, style, and boldness to be specified. You can also request that a report be turned into a data file, or you can export the report in CSV (delimited) format.

INSTALLING AND RUNNING TPL REPORT

Complete instructions for installing and running TPL REPORT are contained in appendixes to this User Manual. The following is a quick summary of the steps to produce reports.

1. Write the codebook statements necessary to describe the data file. Run the codebook processor to create a codebook "object". If you want to make changes to your codebook after it has been processed, you can make the changes and rerun the codebook processor. Otherwise, you only need to do it once. Any number of TPL REPORT jobs can be run using the same codebook object.

2. Write TPL statements to describe the reports you want. Run the procedure to produce the reports. This procedure uses your codebook object and your TPL REPORT statements to read the required data and produce the reports you have described. You can request that the system print the reports immediately, or you can print them later. The reports will always be saved on your disk until you decide to remove them.

3. If the automatic report format is not acceptable for a report, you can reformat it using FORMAT statements in a format request. Write the FORMAT statements to make the desired changes and rerun the report job.

CHAPTER 3

Entering Statements

RULES AND NOTATIONS FOR CODEBOOKS AND REQUESTS

Statement Rules

Codebook, TPL and FORMAT statements are free format; that is, there are no requirements to begin entries at fixed column positions.

When you enter words using lower case letters, TPL REPORT treats them the same as upper case letters unless you enclose them in single or double quote marks (' or ").

For ease of reading, it is best to structure statements so that entries of the same type are neatly aligned.

Identifiers

Identifiers are names that you create to refer to items such as reports and variables. They can be up to 30 characters long and can contain letters, numbers, and the special characters # and underscore (_). An identifier cannot begin with a number and cannot contain embedded blanks. An identifier is terminated by any character from the TPL REPORT character set other than a number, a letter, # or _. Letters can be upper or lower case. When TPL REPORT reads a lower case letter in an identifier, it converts it to upper case.

Values

Numeric values used as constants can contain embedded decimal points. Optional zeros can be added to the left of the values. For example, 053 is the same as 53. Alphabetic values must be enclosed in quote marks. To enter alphabetic values that contain quotes or the backslash character (\), see the instructions below under Print Labels.

Keywords

TPL REPORT uses many words which identify certain functions and must not be used as names. These keywords are shown in an appendix. Keywords can be entered in upper or lower case.

Print Labels

Data names can be given extensive labels which appear automatically on printed output. These labels are bounded with quote marks and are not limited in length. All characters can be used in labels, although the characters ' " and \ require special treatment. Tabs and carriage returns (typed with the **<Enter>** key) should not be used in labels. Tabs are replaced with blanks, and carriage returns are removed when labels are printed.

If you are using single quotes to enclose a label string and need to include a single quote within the string, use two single quotes where you want the single quote to print. An example is **'Inside MSA''s'**, which would print as **Inside MSA's** if used in a report. Similar instructions apply to the use of double quotes.

Backslash

To include the backslash (\) character in a string, enter a double backslash (\\) at the point where you want the backslash to print. This special treatment is necessary because the backslash is used to enter characters that are not on the keyboard.

There are many other label options, all of which are described in a separate chapter called Creating and Formatting Print Labels.

Entering Characters that Are Not on the Keyboard

You may have some characters available on your printer that cannot be entered directly from your keyboard. This is especially true if you want to use special PostScript characters as footnote symbols or if you need to enter non-English language characters with an editor that doesn't support these characters. TPL TABLES provides two ways to enter characters not on your keyboard. One way is to use character names and the other is to use character codes.

You can enter the character as a code by typing **\nnn** (that is backslash followed by 3 decimal digits) to represent the character.

Three digits are always required. If the character can be represented by fewer than 3 digits, add leading zeros. For example, for a character represented by the code 65, enter \065.

The value **nnn** must be the DECIMAL code for the character. The character code tables in some software and printer manuals show the octal or hexidecimal codes for the characters. If you are referring to such a table, you must convert the code to its decimal equivalent. There are tables in the **Character Sets** appendix that show the decimal codes for characters printed in PostScript.

The other way to enter characters not on your keyboard is to use the character name preceded by **&** and followed by **;**. **É** is the character name for the letter **E** with an acute accent. Character names are case-sensitive. **é** is **e** with an acute accent. See the Appendix **Interational** for more on entering characters and the **Character Sets** appendix for a list of supported names.

Mathematical Operators

Statements which involve computations use the mathematical symbols of +(addition), -(subtraction), *(multiplication), /(division), **(exponentiation) and =(equals). Mathematical symbols need not be separated from other elements by spaces.

Comment Entries

You may add your own comments anywhere in a codebook, report request or format request. Comments allow you to include documentation with your statements.

A comment must begin with **/*** and end with ***/.** For example,

```
/* This is a comment. */
```

All characters can be used in comments. The only exception is the pair of characters */, since this pair of characters ends a comment.

Notation Used in Presenting Statement Formats

In this manual, the syntax for TPL REPORT statements is described using a symbolic notation.

- Entries surrounded by [ ] are optional.
- Vertically stacked entries indicate that one entry must be chosen.
- Keywords are presented in UPPER CASE.
- Entries presented in lower case are to be replaced with the proper elements.
- The special delimiters **= ( ) ; > < ^** are presented as they should appear in the statement specification.

The "INCLUDE" Feature

If you have a set of statements or other information that you would like to store in a separate file and then use in multiple codebooks, report requests or format requests, you can use the following notation to get this file included in your codebook or request:

```
%INCLUDE filename
```

TPL REPORT will include the contents of the named file during processing of the codebook or request. You can also use %INCLUDE in a profile.

Some common uses of %INCLUDE are:

1. inclusion of a long list of condition values and labels that apply to more than one control variable, either within the same codebook or in multiple codebooks;

2. inclusion of a long RECODE statement in several report requests;

3. inclusion of a set of FORMAT statements in format requests that apply to a particular group of reports.

The include notation should be at the beginning of a line and on a line by itself. ***NOTE*** *that the include notation must not be followed by a semicolon.*

A codebook, request or profile can have multiple "includes".

"Nesting" is allowed. This means that an included file can include other files. Ten levels of nesting are allowed.

If you have an error in an included file and choose to review your output to find the error message, the review will display the included file as if it is part of the main codebook, request or profile file. Comments will show where the included file begins and ends. If you then wish to edit, you must keep track of which file has the error, because the main file will be transferred to your editor. Do not edit this file, but, instead, bring the appropriate included file into your editor and correct the errors in that file before returning to codebook or request processing.

Example

Assume that we have three variables in the same codebook that all use the same long list of country values and labels. The list can be stored in a file called COUNTRY.LST as follows:

```
(
        'Australia'                     =  01
        / 'Northern Europe' /
        'United Kingdom and Ireland'    =  02
        ' Austria'                      =  03
        ' Belgium'                      =  04
        ' France'                       =  05
                .
                .
                .
        'New Zealand'                   =  38
        'Other Oceania'                 =  39
        / 'N/S'                         =  40
)
```

Then in the codebook, we can reference the list for each country variable:

```
BPF 'Birthplace of father'    CON 2
%INCLUDE COUNTRY.LST

BPM 'Birthplace of mother'    CON 2
%INCLUDE COUNTRY.LST

CIT 'Country of citizenship'  CON 2
%INCLUDE COUNTRY.LST
```

Substitutions for Names, Labels and Numbers

You can make substitutions for identifiers (names), strings and numbers in a codebook, report request, format request or profile. To do this, you assign a name to the item you wish to replace and precede the name with the character **%** (no blanks between). You fill in the specific name, string or number with a **REPLACE** statement.

Note You cannot use REPLACE to substitute for a codebook name.

For example, you might have a set of reports that you produce from time to time and the only thing that you need to change in your report request is the date appearing in the report titles. Rather than looking through your report request to find and change all of the dates to the current date, you would like to make the change in date just once.

To do this type of substitution, you can give the date a name and use this name in all of the report titles with the character **%** in front of the date name. Assuming the date is called MO_YR, you can write a report title such as:

```
'Latest information as of ' %MO_YR '.'
```

Somewhere preceding the first use of MO_YR, you must provide the information to replace MO_YR. For example:

```
REPLACE MO_YR WITH 'January, 1996';
```

For this replacement, the report title will be:

Latest information as of January, 1996.

You will probably want to put your REPLACE statements at the beginning of the request, codebook or profile in which they are used so that they will be easy to find. The only rule with respect to placement is that the **REPLACE** statement for **%name** must always precede the use of **%name**. For example, you cannot use **%name** in a report request and replace it in a format request.

REPLACE statements can be entered at the very beginning or between other statements in requests and in the profile. In a codebook, they can be at the beginning or between entries such as variables or fillers.

To replace a string, you must use quotes in the REPLACE statement. To replace a name or number, just provide the replacement name or number without quotes.

Examples of number replacement

```
REPLACE MASK_TYPE WITH 99,999.99;

COMPUTE INCOME MASK $%MASK_TYPE =
        WEEKLY_INCOME * 52;
```

or

```
REPLACE THIS_MONTH WITH 3;
REPLACE LAST_MONTH WITH 2;

RECODE CURRENT_MONTHS ON MONTH_CODE;
'This month'     IF %THIS_MONTH;
'Last month'     IF %LAST_MONTH;
```

Example of name and label replacement

```
REPLACE SALES_ITEM WITH AUTOS;
REPLACE SALES_LABEL WITH 'Automobiles';

REPORT R1 'Monthly sales figures for ' %SALES_LABEL :
MONTH  DEALERSHIPS  %SALES_ITEM;
```

Putting REPLACE Statements in %INCLUDE Files

If you wish, you can put your REPLACE statements in one or more %INCLUDE files. This means, for example, that you can change entries for a report request without changing the request itself. The %INCLUDE must be entered in the request at a point that precedes the first use of any of the substitutions in the %INCLUDE file.

Example We can redo the preceding example as follows. In a %INCLUDE file called ITEM, we can enter the REPLACE statements:

```
REPLACE SALES_ITEM WITH AUTOS;
REPLACE SALES_LABEL WITH 'Automobiles';
```

In the report request, we can have a %INCLUDE entry:

```
%INCLUDE  ITEM

REPORT R1 'Monthly sales figures for ' %SALES_LABEL :
MONTH  DEALERSHIPS  %SALES_ITEM;
```

With this approach, we can generate reports for different sales items by changing only the file called ITEM to substitute the desired item into the report heading and substitute its description into the report title.

CHAPTER 4

Data and Codebooks

Before requesting reports, you must describe your data file in a codebook so that TPL REPORT will know the names and locations for the variables that you want to use in your reports.

Any data file and codebook that can be used with TPL TABLES can also be used with TPL REPORT. See the following chapters in the TPL TABLES User Manual:

Chapter 5 Data

Chapter 6 Codebook

CHAR Data Type

The character data type of CHAR, entered in the codebook, can be especially useful for reports. A variable described with a type of CHAR is similar to a CONTROL variable, but the possible data values do not need to be listed in the codebook. An example is:

```
INDUSTRY  CHAR  8
```

where the variable called INDUSTRY can have any value that is 8 characters long.

Other examples are:

```
CITY  CHAR  20

PERSON_NAME  CHAR  30
```

CHAR variables can be used in any TPL REPORT statements where control variables can be used. The one exception is the RECODE statement. Only observation and control variables can be recoded.

Tip

If you are using TPL REPORT to create a subfile of your original data using the FORMAT statement called DATA REPORT and you wish to transfer sections of contiguous variables containing character data, you can use the CHAR data type to advantage. Describe the sections as CHAR variables in the codebook and use these CHAR variables in your REPORT statement. The sections of data will be transferred to the new data file output exactly as they appear in the original file. If the entire record consists of character data, *you can transfer the entire record by describing it as a single CHAR variable*.

NOTE: If your records are very long, you may encounter a limit on the size of the CHAR variable. If this is a problem for you, please contact QQQ Software.

The FORMAT statements RETAIN BLANKS; and DELETE BLANKS; can be used to control whether CHAR variables have their leading and trailing blanks stripped when the values are put into reports. The usual default is DELETE, but for DATA REPORTS; the default is RETAIN. Otherwise, you could get misalignment of values because of blanks.

Record Name Variables

Note that record names assigned in the codebook do not have the same meaning when used in a REPORT statement as they do when used in a TABLE statement. If a record name is used in a report, the record number is printed for each record in the report, whereas in TPL TABLES the record name variable has a value of 1 for each record.

Treatment of Data Errors

If there is an error in one or more values of a data record, you will still get a report row for that record. A (d) indicator will be placed in the report cell for any value that has an error. Note that any value is considered valid for a variable of type CHAR, so for this type of variable, any characters that are printable will be displayed in the report.

In the following report, we can see that Employee 1 has a data error in the value for Sex; Employee 2 has a data error in the value for Pay Rate; and Employee 3 has a data error in the value for AGE.

```
REPORT E1 'Report using data with errors.':
        EMPLOYEE  SEX  POSITION  RATE  AGE;
```

Report using data with errors.

Row	Employee	Sex	Type of Job	Pay Rate	AGE
1	1	(d)	Clerk/Typist	5.53	23
2	2	m	Director of Operations	(d)	45
3	3	f	Staff Secretary	7.35	(d)
4	4	f	Executive Secretary	9.21	33
5	5	m	Accounting Clerk	5.00	56

Tip

If we want to see what the actual error values are, we can redefine the variables that have errors and assign a type of CHAR to the redefining variables. This way any value that is printable will show up in our report. For example:

```
EMPLOYEE  'Employee'  RECORD

SEX  'Sex'  CONTROL  1
(
   'Female' = 'f'
   'Male'   = 'm'
)
SEX_CHAR  REDEFINES  SEX  CHAR  2

POSITION  'Type of Job'  CHAR   22

RATE 'Pay Rate' mask 99.99 OBS  7
RATE_CHAR REDEFINES RATE CHAR 7

AGE  CONTROL   2   (16:75)
AGE_CHAR REDEFINES AGE CHAR 2
```

If we use the CHAR versions of the variables in following report, we can see that Employee 1 has an invalid value of 'c' for Sex; Employee 2 has an invalid value of 'a1.49d1' for Pay Rate; and Employee 3 has an invalid value of 'qq' for AGE.

```
REPORT E2 'Report displaying error values as CHAR values':
        EMPLOYEE  SEX_CHAR  POSITION  RATE_CHAR  AGE_CHAR;
```

Report displaying error values as CHAR values

Row	Employee	SEX CHAR	Type of Job	RATE CHAR	AGE CHAR
1	1	c	Clerk/Typist	5.5301	23
2	2	m	Director of Operations	a1.49d1	45
3	3	f	Staff Secretary	7.3501	qq
4	4	f	Executive Secretary	9.2101	33
5	5	m	Accounting Clerk	5.0001	56

CHAPTER 5

Use

ACCESSING THE CODEBOOK

The USE statement must be the first TPL statement in a report request. It takes the form:

Format

```
USE codebookname [ CODEBOOK ];
```

where 'codebookname' is the name assigned to the codebook description of the data file to be accessed by the report request. The keyword CODEBOOK is optional following the codebook name.

The USE statement makes available to all following request statements all names defined in the codebook.

Example

For a codebook named **SURVEY**, the USE statement would be

```
USE SURVEY CODEBOOK;
```

or more simply

```
USE SURVEY;
```

Before the codebook is used, it must be processed to create a file named codebookname.K. In this example, the processed codebook is **SURVEY.K** . TPL REPORT assumes that it is in the directory where you are running your report job.

If the processed codebook is not in the directory where you are running your job, you can enter the path information needed to find the processed codebook. For a codebook named **SURVEY**, the following USE statements provide some examples of acceptable codebook references:

In Windows

```
USE  D:\MY_CBS\SURVEY.K ;
```

In UNIX

```
USE  ../my_cbs/SURVEY.K ;
```

In either of the above examples, the **.K** suffix can be omitted.

Note Comments cannot be inserted in the USE statement between the word USE and the codebook name.

CHAPTER 6

Report

THE REPORT STATEMENT

Introduction

Before requesting reports, you must describe your data file in a codebook so that TPL REPORT will know the names and locations for the data values that you want to use in reports. Details on the types of data files that can be used and how to describe them in a codebook are contained in separate chapters. Note that if you have already prepared a codebook for TPL TABLES, you can use this codebook with TPL REPORT.

REPORT statements are prepared using an editor and saved in a file called a TPL report request. The smallest report request contains just a USE statement, telling TPL REPORT which codebook to use, and a REPORT statement.

The REPORT statement assigns a name to a report and specifies the variables whose values will be listed for each record included in the report. The order of the variables in the list determines the **order of the columns** in the report output. A SORT statement can be used to control the **order of the rows**. If no SORT statement is used with the REPORT statement, the rows will be reported **in the same order as the records** in the data file. The SORT statement is described in detail in a separate chapter.

Multiple REPORT statements can be included in the same job. A separate output file is created *for each* REPORT statement. A report file name begins with the **report-name** assigned in the REPORT statement and ends with the suffix **.rep** . For example, if the report-name is IND_REPT, the report is saved in a file called **ind_rept.rep**.

Description of the REPORT Statement

Format REPORT report-name : var1 [var2 var3 .....] ;

The **report-name** is restricted to 8 characters in length. The variables in the list can be separated by one or more blanks, a comma or the word **THEN**.

Examples

```
REPORT REP1 : RECORD_ID ENTERPRISE PRODUCT_CODE SALES ;

REPORT REP2 : ENTERPRISE, PRODUCT_CODE, SALES ;

REPORT SALESREP : RECORD_ID THEN ENTERPRISE THEN SALES;
```

All variables in the variable list must be in the codebook or otherwise already defined, for example in a COMPUTE statement.

Using Record Names and Built-in Variables

If you use a **record name** from the codebook in a REPORT statement, the value shown in the report output will be the **record number** for the named records. This is different from the treatment of record name in TPL TABLES where the value is 1 for each record.

The built-in observation variable **COUNT** contains the current record number. For a flat file, COUNT has the same value as the record name variable. For a hierarchical file, COUNT and record name variables have different meanings. See the "Hierarchies" chapter for details.

The built-in control variable **TOTAL** that exists in TPL TABLES can also be used in TPL REPORT. However, since it has a value of 1 for each record, it is not very interesting if used in a report.

OTHER, ALL and EXCEPT in the Variable List

OTHER and ALL

The word **OTHER** can be used in the variable list and means all variables in the codebook not previously included in the variable list. Since OTHER only refers to codebook variables, you must explicitly include computed variables in your variable list if you want them in the report.

The word **ALL** can be used in the variable list and means all variables in the codebook whether previously referenced in the list or not. As with OTHER, ALL only refers to codebook variables, so you must explicitly include computed variables in your variable list if you want them in the report.

The output column order for variables included in the ALL or OTHER categories is in alphabetical order by variable name. If you want a different order for the columns, you must list the variables explicitly in the REPORT statement.

EXCEPT

The phrase **EXCEPT var1 var2 ...** can be included at the end of the variable list to delete the variables **var1 var2 ...** from the report if they are already included in the variable list. If any variable following EXCEPT has not already been included in the report, EXCEPT will have no effect for that variable. TPL REPORT will not issue a warning message to tell you this.

As with other variables in the REPORT statement, variables that follow EXCEPT must already be described in the codebook or created by a previous statement such as COMPUTE.

The following REPORT statement displays values for all variables in the codebook except RECORD_ID and REGION.

Example

```
REPORT SALESREP : ALL EXCEPT  RECORD_ID REGION;
```

The word **EXCLUDE** can be used instead of **EXCEPT** to mean the same thing.

Example

```
REPORT ONE:  ALL EXCLUDE AGE SEX;
```

Report Output Format

This section describes the automatic report formatting. Many FORMAT statements are available for changing the format defaults. See the FORMAT chapter for details.

Tip

If you have a large data file, your reports may be very large also. To easily check the format of your reports, use a SELECT statement to choose only a few records of data. The option that lets you select a specific number of records provides a good way to do this. Working with only a small number of records, you can quickly review your results and adjust such things as column width, using the FORMAT statement called COLUMN WIDTH, until you get the format that is best for you. Then you can remove the SELECT statement from your report request and prepare your reports with the complete data file. See the SELECT chapter for details.

Basic Format

The report output contains a column of data for each variable included in the REPORT statement. Each column is headed by its variable name or label at the beginning of each page of the report. If a report is too wide to fit on one page, it is automatically broken into as many pages as required to hold all of the columns. Each page of the report is centered.

The NUMBER Variable

NUMBER is a special built-in variable in TPL REPORT. It contains **the row number** for each row of data in a report. By default, a NUMBER column is included as the first column for each page of report output, headed by the label "Row". If you have a wide report that is broken into sections (called banks) on multiple pages, you can use the row numbers to match the data rows from page to page.

The NUMBER column can be referenced as a variable name in FORMAT statements. The format options for NUMBER include moving the column, deleting it and changing the label. For example:

FOR VARIABLE NUMBER: DELETE COLUMN;

Column Widths

The **default width** for each column is based on a combination of factors and designed to be a compromise between legibility and the amount of data that can be shown within the page width. The rules are summarized below. In all cases, the width includes one extra character to allow for a blank space between columns. The minimum column width that will be assigned automatically is is 4.

The **optimal** column width for any variable depends on its contents and the requirements of your particular report. Changing the column width is straightforward and simple with the FORMAT statement called COLUMN WIDTH. We recommend that you use this statement if the default width is not appropriate for your report.

The Defaults

For **CONTROL and CHAR** variables described in the codebook, if the width of the variable name is less than the field size, the column width is the same as the field size plus 1 up to a maximum width of 15. If the variable name is longer than the field size, the width is expanded to the length of the variable name plus 1.

For variables created by **RECODE** statements, the column width is the length of the variable name plus 1.

For **OBSERVATION** variables, including those computed in the report request, the column width is 10, unless the variable name is longer. In that case, the column width is the width of the variable name plus 1.

For the **NUMBER** variable, the column width is 10. Note that if you use the FORMAT statement RETAIN SIDE RULES, the first position of the NUMBER column is used for the left side rule.

Display Format for Data Values

For **CHAR** variables, the data values are displayed exactly as found in the data file, except that leading and trailing blanks are removed for printed reports. If you use the FORMAT statement DATA REPORTS; to create a data file output, the blanks will be retained by default. If you wish to change the default treatment for leading and trailing blanks, see the FORMAT statement called RETAIN (or DELETE) BLANKS.

For **CONTROL** variables, the data values are displayed exactly as found in the data file, except that leading and trailing blanks are always removed by default. If you wish to retain the blanks for a CONTROL variables, you can RECODE it, then use the new recoded version in your report. For example:

```
RECODE LNAME_WITH_BLANKS ON LNAME;
VALUE IF ALL;
```

For variables created by **RECODE** statements, the values are displayed exactly as specified.

For **OBSERVATION** variables, any non-character values, such as binary or floating point, are converted to character before printing. Print masks are used to format the data for any variables that have masks specified in the codebook or in a COMPUTE statement. The default print mask is integer (i.e. no decimal places) with commas as the thousands separator. See the chapter called "Masks" for details on other data formats.

Wide Values

Any value that is too large to print in the space alloted for its column is broken into as many lines as required to show the value within the width of the column.

Alignment

By default, values are centered. You can change the alignment of OBSERVATION values by specifying a print mask that includes an alignment such as LEFT or RIGHT. You can change the alignment for all types of values, including CONTROL and CHAR, with the FORMAT statement called ALIGN COLUMNS.

Special Indicators in Data Cells

A report **cell** is a location defined by the intersection of a row and a column. Normally, a data value is displayed in each report cell.

Sometimes an exact data value cannot be displayed in a cell. When this happens, TPL REPORT inserts a special indicator in place of the value. The indicators can be any of the following:

(c) Computation error

(d) Data error

(f) Value does not fit

(n) Value is null

Titles and Labels

See the chapter called "Labels" for details on the many options available for titles and other labels.

Each report page begins with the **report title**. If you have not included a title in the REPORT statement, the report name will be used as the title.

Column labels are taken from the variables. If a variable does not have a label, the variable name is used.

Wide Labels

Any title or other label that is too wide for the available space is broken into multiple lines.

Sample REPORT Request and Report Output

For this example, we will use the following data file and codebook.

Data File

```
f       Clerk/Typist            5.53
m       Director of Operations  21.49
f       Staff Secretary         7.35
f       Executive Secretary     9.21
m       Accounting Clerk        5.00
f       Payroll Clerk           5.19
f       Accounting Supervisor   8.65
```

Codebook

```
BEGIN JOBS CODEBOOK

EMPLOYEE 'Employee' RECORD

SEX 'Sex' CONTROL  1
(
   'Female' = 'f'
   'Male'   = 'm'
)

FILLER 5

POSITION 'Type of Job' CHAR  22

FILLER 5

RATE 'Pay Rate' mask $99.99  OBS  5

END JOBS CODEBOOK
```

Report Request

```
USE JOBS CODEBOOK;

REPORT R1:  SEX  POSITION;

REPORT R2:  ALL;

REPORT R3:  ALL EXCEPT SEX;
SORT R3 ON RATE(D);
```

Report Output

The sample REPORT request produces three reports, one for each statement. Each report is stored in a separate file with a name built from the report name. The report file names are **r1.rep**, **r2.rep** and **r3.rep**.

The first two reports are in the order of the data file since there are no associated SORT statements.

r1.rep

The REPORT statement is:

```
REPORT R1:  SEX  POSITION;
```

The report begins with the NUMBER column that contains the report row numbers and the label "Row". The next two columns are SEX and POSITION (with the label "Type of Job"), in the order requested by the REPORT statement. Note that where the job types are too wide for the default column size, they are automatically wrapped in the column.

R1

Row	Sex	Type of Job
1	f	Clerk/Typist
2	m	Director of Operations
3	f	Staff Secretary
4	f	Executive Secretary
5	m	Accounting Clerk
6	f	Payroll Clerk
7	f	Accounting Supervisor

r2.rep

The REPORT statement is:

```
REPORT R2:  ALL;
```

The report contains the NUMBER column. Since the REPORT statement requests ALL variables, the NUMBER column is followed by a column for each variable described in the codebook. These columns are in alphabetical order of the variable names, EMPLOYEE, POSITION, RATE and SEX.

Note that since EMPLOYEE is the record name, its value is the record number. Since all records are displayed, and the order is the same as the data file order, the values in the EMPLOYEE column match the values in the NUMBER column.

R2

Row	Employee	Type of Job	Pay Rate	Sex
1	1	Clerk/Typist	5.53	f
2	2	Director of Operations	21.49	m
3	3	Staff Secretary	7.35	f
4	4	Executive Secretary	9.21	f
5	5	Accounting Clerk	5.00	m
6	6	Payroll Clerk	5.19	f
7	7	Accounting Supervisor	8.65	f

r3.rep

The last report is sorted in descending order by salary rate as specified by the corresponding SORT statement. Since the report is sorted, the report order is differenct from the order of the data file and the values in the NUMBER and EMPLOYEE columns do not match.

```
REPORT R3:  ALL EXCEPT SEX;
SORT R3 ON RATE(D);
```

R3

Row	Employee	Type of Job	Pay Rate
1	2	Director of Operations	21.49
2	4	Executive Secretary	9.21
3	7	Accounting Supervisor	8.65
4	3	Staff Secretary	7.35
5	1	Clerk/Typist	5.53
6	6	Payroll Clerk	5.19
7	5	Accounting Clerk	5.00

CHAPTER 7

Sort

SORTING REPORTS

The SORT statement lets you choose a specific order for your report output that is based on the values of particular variables. If you do not have a SORT statement for a report, the rows of the report are listed in the order of the data file.

The SORT statement references a particular REPORT statement. It lists one or more variables and the sort order desired for each.

Format

```
SORT [FOR] [REPORT] report-name ON var1(a) var2(d) ..... ;
```

where the "a" or "d" in parentheses are optional choices of sort order. If they are not included, the sort order is ascending. Use an "**a**" to indicate **a**scending order (the default order) or a "**d**" to indicate **d**escending order for a particular variable.

In general, the definitions of "ascending" and "descending" will be what you expect, but you should note that for variables of type CONTROL, the order will match the display order given in the codebook. The default display order is "DISPLAY AS LISTED". This means that the values are sorted in the order of their listing in the codebook. Thus, if 'z' is listed before 'a', an ascending order in the SORT statement will result in 'z' rows appearing before 'a' rows. If you want TPL REPORT to put the values for this type of variable into sort sequence rather than listed sequence, you must add the clause "DISPLAY AS SORTED" in the codebook description of that variable.

The variables in the SORT statement need not be the same as the variables named in the corresponding REPORT statement.

Variables created with RECODE statements cannot be used in a SORT statement.

Example

```
REPORT REP1 :  REGION  RECORD_ID  ENTERPRISE
                       PRODUCT_CODE  SALES ;

SORT  REPORT REP1  ON  REGION(d)  PRODUCT_CODE;
```

In this example, the sort order for the report follows the order of the variables in the SORT statement. The first variable determines the primary sort order, the second variable determines the secondary order, and so on. In the above example, the report is sorted by REGION in descending order and, within REGION, sorted by PRODUCT_CODE in ascending (the default) order.

If a SORT statement is used, only the variables named in the SORT statement will affect the order of the report rows. Values for other variables will not cause a predictable order. For example, if you name SEX as the sort variable where SEX has values 1 and 2, all rows with SEX value 1 will be sorted together and all rows with SEX value 2 will be sorted together, but within either SEX the row order cannot be predicted.

Example

In the following shipping report, the rows are sorted by dollar amount of shipment from the highest to the lowest (descending order) within type of shipping service (Tanker, Tramp, ...).

```
REPORT S1 'Shipping report sorted by type of service and dollar value':
TYPE_SERVICE THEN DOLLARS THEN STONS THEN NEW COUNTRY;

SORT REPORT S1 ON TYPE_SERVICE DOLLARS(D);
```

Shipping report sorted by type of service and dollar value

Row	Type of Service	Dollars	Short Tons	Country
1	Tanker	8,948,210	48,369	201 MEXICO
2	Tanker	7,584,000	33,841	225 PANAMA
3	Tanker	841,260	2,208	201 MEXICO
4	Tanker	820,913	22,048	241 JAMAICA
5	Tanker	809,327	1,332	201 MEXICO
6	Tanker	651,349	18,866	122 CANADA
7	Tanker	504,584	1,328	241 JAMAICA
8	Tanker	282,986	4,173	219 NICARAGUA
9	Tanker	192,867	3,717	247 DOMINICAN REPUBLIC
10	Tramp	1,976,939	12,107	122 CANADA
11	Tramp	1,869,970	15,383	241 JAMAICA
12	Tramp	577,745	3,300	247 DOMINICAN REPUBLIC
13	Tramp	517,371	2,756	215 HONDURAS
14	Tramp	444,049	3,703	205 GUATEMALA
15	Tramp	348,339	818	201 MEXICO
16	Tramp	197,046	1,358	225 PANAMA
17	Tramp	165,718	4,510	245 HAITI
18	Tramp	97,768	2,629	201 MEXICO
19	Tramp	3,962	2	211 EL SALVADOR

TOP n Option for SORT

You can include one **TOP n** specification in the variable list of a SORT statement to filter the report so that it includes only records that belong to the top categories.

Format

SORT [FOR] [REPORT] report-name ON [var1] TOP-n-clause [var2 .....] ;

The format for the **TOP-n-clause** is:

TOP n rep-var RANKED ON obs-var

where **rep-var** can be any variable, **obs-var** can be any observation variable, and **n** is an integer. Normally, you will want to use only variables that are included in the corresponding REPORT statement, since the results will be difficult to interpret otherwise. The clause can be interpreted as, "Within each value of rep-var, aggregate the obs-var values and report all records in the top n categories of rep-var."

Example

```
REPORT REP1 'Top Sales Regions' :  REGION  SALES ;

SORT REP1  ON  TOP 3 REGION RANKED ON SALES ;
```

In this example, TPL REPORT will find the regions with the top 3 total sales and report all records in those regions.

Following are the REGION and SALES values for 11 records:

Northeast	50000
Northeast	50000
Northwest	150000
Northwest	50000
Northcentral	250000
Southeast	**100000**
Southeast	**100000**
Southeast	**100000**
Southwest	**200000**
Southwest	**200000**
Southcentral	**350000**

After aggregating the SALES amounts for all of the records in each REGION, we find that the top 3 regions are the ones in the South.

Northeast	100000
Northwest	200000
Northcentral	250000
Southeast	**300000**
Southwest	**400000**
Southcentral	**350000**

All of the records in these three regions will be included in the report and listed in order of REGION as follows:

Top Sales Regions

Row	REGION	SALES
1	Southcentral	350,000
2	Southeast	100,000
3	Southeast	100,000
4	Southeast	100,000
5	Southwest	200,000
6	Southwest	200,000

We can also request sales subtotals for the top 3 regions, as shown in the next example. For complete details on totals and subtotals, see the chapter called "Totals".

```
REPORT REP2 'Top Sales Regions with Sales Totals':   REGION  SALES;
SORT REP2 ON TOP 3 REGION RANKED ON SALES;

SUBTOTAL SALES_TOTAL 'Total Sales for Region' ON
        REGION  REPORT REP2;
DISPLAY SALES;
```

Top Sales Regions with Sales Totals

Row	REGION	SALES
1	Southcentral	350,000
Total Sales for Region		350,000
2	Southeast	100,000
3	Southeast	100,000
4	Southeast	100,000
Total Sales for Region		300,000
5	Southwest	200,000
6	Southwest	200,000
Total Sales for Region		400,000

Note

If there is a tie between categories (regions in this example), the number of regions chosen will still be limited to 3. In the following, we have a tie between Northeast and Southeast. Only one of these two regions will be included in the top 3.

Northeast	**300000**
Northwest	200000
Northcentral	250000
Southeast	**300000**
Southwest	**400000**
Southcentral	**350000**

If there are were fewer than 3 regions represented in the data, for example only 2, then only the records from the 2 regions could be reported.

There can be only one TOP-n-clause in a SORT statement, but it can be inserted anywhere in the variable list. The placement determines the number of categories for which records will be reported according to ranking. For example, we can take the REPORT and SORT statements above and insert PRODUCT_CODE as follows:

```
REPORT REP1 'Top Sales Regions' :  PRODUCT_CODE  REGION  SALES ;

SORT REP1  ON  PRODUCT_CODE
       TOP 3  REGION RANKED ON SALES;
```

We will get the TOP 3 regions ranked by sales within each PRODUCT_CODE.

To Get the Bottom-Ranked Records Instead of the Top

If you want to get the records that rank at the bottom rather than the top, you should compute a new observation variable for the ranking using negative values. Then the new variable can be used in the TOP n clause to get the lowest-ranked records. The original variable should be used in the REPORT statement so that the original, positive values will be displayed.

Example

```
COMPUTE NEG_SALES  =  - SALES;

REPORT REP1_N 'Lowest-ranked  Sales Regions':  REGION  SALES;
SORT REP1_N  ON  TOP 3 REGION RANKED ON NEG_SALES;
```

Record Names, COUNT and TOTAL

When a **record name** from the codebook is used in a report, the record number is displayed for each record in the report. There may be cases in which it is useful to reference a record name as the **rep-var**. Assume that we wish to see 4 records with top values for SALES, regardless of any other characteristic. Assume also that the record name in the codebook is SALES_RECORD. Then we could specify:

```
REPORT REP3 'Top 4 Sales Records':  SALES_RECORD  REGION  SALES;

SORT REP3  ON  TOP 4  SALES_RECORD RANKED ON SALES;
```

The report will then include only 4 records listed in record order:

Top 4 Sales Records

Row	SALES RECORD	REGION	SALES
1	5	Northcentral	250,000
2	9	Southwest	200,000
3	10	Southwest	200,000
4	11	Southcentral	350,000

Note that if there are more than 4 records with the top sales value, only 4 of them will be shown in the report, because each record number is unique.

In a file with only one type of record (non-hierarchical), the built-in variable **COUNT** can be used in the TOP n clause with the similar results, since the value of COUNT is the same as the record number when there is only one type of record.

If the built-in variable **TOTAL** is used in the TOP n clause, the results will be meaningless.

Note on TOP n and Negative Values

When you request, for example, TOP 3 X RANKED ON Y, the algebraic sum of Y values is used to determine its ranking. Thus, if values for Y are negative, a sum of 0 could be the top ranked value.

If you want the sum of the absolute values to determine the ranking, then you should compute the sum of absolute values:

```
COMPUTE NEW_Y = ABS(Y);
```

Use the NEW_Y variable in place of the Y variable in your TOP n clause.

CHAPTER 8

Totals

THE SUBTOTAL AND GRAND TOTAL STATEMENTS

Subtotals

Subtotals can be displayed for sorted reports. To request subtotals, use a SUBTOTAL statement with a DISPLAY clause. The SUBTOTAL statement tells where a subtotal should be inserted; the DISPLAY clause tells what should be subtotalled, i.e. for which variables you want the subtotals to be displayed.

The SUBTOTAL Statement

Format

```
SUBTOTAL subtotal-name [subtotal-label] ON sort-var
        [FOR] REPORT report-name;
```

Each SUBTOTAL statement applies to the single sorted report referenced by **report-name**. The **sort-var** named in the statement can be any variable named in the SORT statement for the report. A subtotal row will be inserted whenever there is a change in value for that variable from one row to the next.

The **sort-var** can be an OBSERVATION, CONTROL or CHAR variable. Only one variable can be named in a single SUBTOTAL statement. If you want to have subtotals inserted at more than one location is a report, use an additional SUBTOTAL statement for each location.

The **subtotal-name** is required. The **subtotal-label** is optional. The **default label is 'Subtotal'**.

The DISPLAY Clause

Format

```
DISPLAY    ALL                                                    ;
           obs-var-1 [MASK mask] [ .... obs-var-n [MASK mask]]    ;
```

Each SUBTOTAL statement needs a DISPLAY clause. With DISPLAY, you name the variables whose subtotals you want to display in the subtotal rows.

1. **DISPLAY ALL;** gives subtotals for all observation variables named in the REPORT statement. If ALL variables are in the report, then subtotals are displayed for all of the observation variables.

2. DISPLAY can choose **specific observation variables** from those named in the REPORT statement. If the REPORT statement specifies ALL variables, then any of the observation variables can be chosen.

How Subtotals Are Displayed

Subtotals are displayed as separate lines without row numbers. The subtotal labels are displayed in the NUMBER column if it is present. If the NUMBER column has been deleted, there will be no labels for the subtotal lines. Each subtotal line immediately follows a detail report row. If the subtotal label is broken into multiple lines, the subtotal values will be on the same line as the first line of the label. The next detail row follows immediately after the subtotal line, or the last line of the subtotal label, if present.

For each observation variable chosen by the DISPLAY statement, there will be a subtotal entry in that variable's column on the subtotal lines. If the report is banked, each bank will include subtotal lines, and labels if present, but the subtotals will only appear in the bank(s) where the DISPLAY variables occur.

Example

```
REPORT T1    'Shipping report by month and country with subtotals '
             'for each month':

MONTH THEN COUNTRY THEN DOLLARS THEN TONS;
SORT T1 ON MONTH COUNTRY;
SUBTOTAL MONTH_TOT ON MONTH  FOR REPORT T1;
   DISPLAY  DOLLARS MASK '$'9,999,999  TONS mask 99,999;
```

The report is sorted by MONTH and COUNTRY. Monthly subtotals are calculated for DOLLARS and TONS. These subtotals are displayed following the last row for each MONTH. Since no label is included in the SUBTOTAL statement, the subtotal lines have the default label 'Subtotal'.

Shipping report by month and country with subtotals for each month

Row	MONTH	COUNTRY	Dollars	Short Tons
1	01	201 MEXICO	63,282	5,577
2	01	201 MEXICO	573,804	14,584
3	01	201 MEXICO	833,647	1,657
4	01	247 DOMINICAN REPUBLIC	500,206	2,965
5	01	247 DOMINICAN REPUBLIC	560,028	3,199
6	01	283 FRENCH WEST INDIES	323,471	5,943
Subtotal			$2,854,438	33,925
7	02	122 CANADA	651,349	18,866
8	02	201 MEXICO	31,516	2,638
9	02	211 EL SALVADOR	1,280,167	7,200
10	02	215 HONDURAS	110,000	1,550
11	02	225 PANAMA	245,882	1,523
12	02	241 JAMAICA	12,990,192	53,680
13	02	241 JAMAICA	1,437,435	6,653
14	02	245 HAITI	238,708	1,548
15	02	245 HAITI	21,937	607
Subtotal			$17,007,186	94,265
16	03	122 CANADA	1,541,819	9,590
17	03	201 MEXICO	841,260	2,208
18	03	201 MEXICO	776,657	1,284
19	03	201 MEXICO	407,641	4,077
20	03	201 MEXICO	8,948,210	48,369
21	03	211 EL SALVADOR	567,294	3,466
22	03	225 PANAMA	23,487	1,958
23	03	241 JAMAICA	7,554,538	37,813
Subtotal			$20,660,906	108,765
24	04	122 CANADA	1,793,309	30,219
25	04	201 MEXICO	809,327	1,332
26	04	201 MEXICO	1,498,450	4,543
27	04	205 GUATEMALA	55,230	331
28	04	241 JAMAICA	675,317	2,451
Subtotal			$4,831,633	38,876
29	05	201 MEXICO	389,504	5,607
30	05	205 GUATEMALA	1,403	3
31	05	211 EL SALVADOR	9,968	19
32	05	225 PANAMA	7,584,000	33,841
Subtotal			$7,984,875	39,470

Grand Totals

The GRAND TOTAL statement is a simple version of the SUBTOTAL statement. You do not need to specify the location for the grand total or sort the report, because grand totals are calculated for the entire report and placed at the end.

Format

```
GRAND TOTAL gt-name [gt-label]  [FOR] REPORT report-name ;

DISPLAY      ALL                                                    ;
             obs-var-1 [MASK mask] [ .... obs-var-n [MASK mask]]    ;
```

Note that GRAND TOTAL can be two words or one, as in GRANDTOTAL.

The **gt-name** is required. The **gt-label** is optional. The **default label is 'Total'**.

Each GRAND TOTAL statement applies to a single report.

Each GRAND TOTAL statement must include a DISPLAY clause. With DISPLAY, you name the variables whose totals you want to display at the end of the report.

1. DISPLAY can choose **specific observation variables** from those named in the REPORT statement. If the REPORT statement specifies ALL variables, then any of the observation variables can be chosen.

2. **DISPLAY ALL;** gives totals for all observation variables named in the REPORT statement. If ALL variables are in the report, then totals are displayed for all of the observation variables.

Example

```
REPORT T2 'Shipping report with grand totals':
MONTH THEN COUNTRY THEN DOLLARS THEN TONS;
SORT T2 ON MONTH COUNTRY;
GRAND TOTAL GT  FOR REPORT T2;
DISPLAY ALL;
```

For REPORT T2, totals are calculated for **all** observation variables and displayed at the end of the report with the label 'Total'.

Shipping report with grand totals

Row	MONTH	COUNTRY	Dollars	Short Tons
1	01	122 CANADA	945,716	5,846
2	01	201 MEXICO	348,339	818
3	01	201 MEXICO	63,282	5,577
4	01	201 MEXICO	573,804	14,584
5	01	205 GUATEMALA	1,268,846	7,767
6	01	215 HONDURAS	210,000	2,868
7	01	219 NICARAGUA	282,986	4,173
8	01	225 PANAMA	33,427	2,607
9	01	247 DOMINICAN REPUBLIC	560,028	3,199
10	02	201 MEXICO	686,947	1,803
11	02	219 NICARAGUA	25,259	660
12	02	241 JAMAICA	1,437,435	6,653
13	03	122 CANADA	1,458,515	9,514
14	03	201 MEXICO	8,948,210	48,369
15	03	201 MEXICO	407,641	4,077
16	03	225 PANAMA	1,686,577	7,899
17	03	241 JAMAICA	164,851	4,446
18	03	247 DOMINICAN REPUBLIC	577,745	3,300
19	04	201 MEXICO	1,441,891	3,252
20	04	201 MEXICO	1,498,450	4,543
21	04	201 MEXICO	809,327	1,332
22	04	247 DOMINICAN REPUBLIC	192,867	3,717
23	04	247 DOMINICAN REPUBLIC	230,719	1,652
24	05	201 MEXICO	97,768	2,572
25	05	205 GUATEMALA	5,905	6
Total			23,956,535	151,234

Subtotals and Grand Totals in the Same Report

The same report can contain both subtotals and grand totals. In the following example, subtotals and a total are calculated for all observation variables in the report.

```
REPORT T3 'Shipping report with subtotals and grand total':
MONTH THEN COUNTRY THEN DOLLARS THEN TONS;

SORT T3 ON MONTH COUNTRY;

SUBTOTAL MONTH_TOT 'Month subtotals' ON MONTH FOR REPORT T3;
        DISPLAY ALL;
SUBTOTAL COUNTRY_TOT 'Country subtotals' ON COUNTRY
        FOR REPORT T3;
        DISPLAY ALL;
GRAND TOTAL GT  FOR REPORT T2;
        DISPLAY ALL;
```

Shipping report with subtotals and grand total

Row	MONTH	COUNTRY	Dollars	Short Tons
1	01	201 MEXICO	855,715	1
2	01	201 MEXICO	348,339	818
3	01	201 MEXICO	1,564,748	5,111
4	01	201 MEXICO	573,804	14,584
Country subtotals			3,342,606	20,514
5	01	219 NICARAGUA	282,986	4,173
Country subtotals			282,986	4,173
6	01	241 JAMAICA	5,360,348	37,482
Country subtotals			5,360,348	37,482
7	01	247 DOMINICAN REPUBLIC	560,028	3,199
Country subtotals			560,028	3,199
Month subtotals			9,545,968	65,368
8	02	122 CANADA	753,536	7,223
Country subtotals			753,536	7,223
9	02	223 COSTA RICA	256,739	1,608
Country subtotals			256,739	1,608
10	02	245 HAITI	85,025	484
Country subtotals			85,025	484
Month subtotals			1,095,300	9,315
11	03	122 CANADA	1,458,515	9,514
Country subtotals			1,458,515	9,514
12	03	201 MEXICO	407,641	4,077
Country subtotals			407,641	4,077
13	03	205 GUATEMALA	1,623,436	8,373
14	03	205 GUATEMALA	769,402	5,069
Country subtotals			2,392,838	13,442
15	03	225 PANAMA	1,686,577	7,899
Country subtotals			1,686,577	7,899
16	03	241 JAMAICA	7,554,538	37,813
Country subtotals			7,554,538	37,813
17	03	247 DOMINICAN REPUBLIC	577,745	3,300
Country subtotals			577,745	3,300
Month subtotals			14,077,854	76,045
18	04	247 DOMINICAN REPUBLIC	230,719	1,652
Country subtotals			230,719	1,652
Month subtotals			230,719	1,652
19	05	122 CANADA	749,796	19,243
Country subtotals			749,796	19,243
20	05	201 MEXICO	97,768	2,572
Country subtotals			97,768	2,572
21	05	225 PANAMA	7,584,000	33,841
Country subtotals			7,584,000	33,841
Month subtotals			8,431,564	55,656
Total			33,381,405	208,036

Use of Record Names and COUNT in Subtotals and Grand Totals

If your use either a record name from the codebook or the built-in variable COUNT as display variables for totals, you will get the count of records contributing to the totals.

In the following report request, both the record name and COUNT are used as display variables for a subtotal based on the variable Sex.

```
USE jobs CODEBOOK;

REPORT tr 'Record name and count in subtotals.':
        sex employee COUNT rate;

SORT tr ON sex;

SUBTOTAL st ON sex FOR REPORT r;
DISPLAY employee COUNT rate;
```

Record name and count in subtotals.

Row	Sex	Employee	Count	Pay Rate
1	f	6	6	5.19
2	f	1	1	5.53
3	f	3	3	7.35
4	f	7	7	8.65
5	f	4	4	9.21
Subtotal		5	5	35.93
6	m	2	2	21.49
7	m	5	5	5.00
Subtotal		2	2	26.49

The record name is Employee. As you can see in the report, the subtotal values for Employee and COUNT are the same. The subtotal for Sex = 'f' has 5 records contributing, while the subtotal for Sex = 'm' has 2 records contributing.

Referencing Subtotals and Grand Totals in FORMAT Statements

SUBTOTAL and GRAND TOTAL names can be referenced in FORMAT statements, for example. to replace labels. Use:

FOR VARIABLE subt-name: action; or

FOR VARIABLE gt-name: action;

Example FOR VARIABLE sex_subt: REPLACE LABEL WITH ‘Sex Subtotal’;

CHAPTER 9

Select

SELECTING SUBSETS OF THE DATA

The SELECT statement specifies conditions that must be met by each record of the data file to qualify for inclusion in the reports.

The SELECT statement applies to all reports within the request and normally appears immediately after the USE statement. If more than one SELECT statement is used in the request, the conditions in all SELECT statements must be met for a record to be included in the reports.

Note that if your data file is hierarchical, the unit to be selected will be a hierarchical unit rather than a single record. The chapter on processing hierarchical files contains additional information on this subject.

Data can be selected based on data values, or certain sections or percentages of the data can be selected.

Selection Based on Data Values

Format This type of selection takes one of the following two forms:

```
SELECT   IF          condition1   [AND condition2.....];
         UNLESS                    OR

                          or,

SELECT   IF          NOT ( condition1    [ AND condition2...] );
         UNLESS                          OR
```

Examples

```
SELECT  IF  ACCOUNT  NOT = 24765;

SELECT  IF ACCOUNT IS NOT EQUAL TO 24765;

SELECT  UNLESS  (REGION = 'A1' AND INCOME <= 12000);

SELECT  IF  (INCOME >= 12000 AND INCOME < 20000) OR
          (STATE = MONTANA AND
          (OCCUPATION = 1 OR OCCUPATION = 5));

SELECT  IF  OCC_CODE IN (2000, 2001, 2002, 2003, 3000)
         AND INCOME >= 30000;

SELECT  IF OCCUPATION = ACCOUNTANT AND
         NOT (REGION = 'D3' OR REGION = 'E5');

SELECT  UNLESS (STATE='CA' AND
         (JOB_BANK=10 OR JOB_BANK=20))
         OR (STATE='IL' AND (JOB_BANK=13 OR
         JOB_BANK=15));
```

When the IF option is used, all records meeting the conditions will be selected. When the UNLESS or IF NOT option is used, all records which do not satisfy the conditions will be selected. If no SELECT statement appears in the request, all records will be included in the reports, regardless of their characteristics.

A condition expresses a relationship or a set of values. It can include codebook variables, computed variables, literal values, and arithmetic expressions.

Relations can be expressed by either symbols or words. The words IS and TO are optional. Following is a list of the relation symbols and their English equivalents:

Symbol	English expression
<	[IS] LESS THAN
>	[IS] GREATER THAN
=	[IS] EQUAL [TO] EQUALS
^<	[IS] NOT LESS THAN
^>	[IS] NOT GREATER THAN
^=	[IS] NOT EQUAL [TO]
<=	[IS] LESS THAN OR EQUAL [TO]
>=	[IS] GREATER THAN OR EQUAL [TO]

Types of Conditions

A condition can test for a **relationship**, or it can test a variable to see if it has any of the values specified in **a set of values**. We first describe how relationships can be tested, then follow with a section on sets of values.

Relationships

A condition that tests a relationship can take several forms. In each form which follows, **re** stands for a relation.

- **Comparing a variable to a value**

Format

```
variable re literal-value
```

In this type of condition, the **variable** is compared to a **literal-value**.

Examples

```
SELECT IF AMOUNT < 100.75;
SELECT IF REGION = 'A1';
```

In the first example, the statement would cause selection of all records with an AMOUNT less than 100.75. In the second example, selection would be of all records containing the value 'A1' for the control variable REGION.

The **variable** in the condition can be any codebook or computed variable, and **literal-value** can be a number or a character string.

If the variable is an **observation** variable, the value can be a number, with or without a sign. The number can contain a decimal point but no commas.

If a **control** variable is compared to a literal value that is not all numeric, the value must be surrounded by quote marks.

If the variable is a **char** variable, the value must be surrounded by quote marks and must include any leading blanks or zeros. The only relations that should be used with char variables are equal '=' and not equal '^='. If other relations are used, the results are unpredictable. For example, if LAST_NAME is a char variable,

Example

```
SELECT IF LAST_NAME = 'Smith' ;
```

will successfully select all records with a value of 'Smith' for LAST_NAME if the value 'Smith' is left-aligned, i.e. not preceded by blanks in the data records.

When the same variable is compared with two or more distinct values, the subject must be repeated each time. For example:

Example SELECT IF AGE=14 OR AGE=18 OR AGE=29;

For a simplified way of expressing this type of condition, see the section on **Sets of values**.

If the "DISPLAY AS SORTED" clause is not used with a codebook **control** variable, all references to a range of values expressed in a SELECT must be based on the order of the conditions listed, rather than the sorted sequence of the condition values. For example, consider the following control variable entry:

```
REGION CON 1
 (
        SW = 'D'
        NW = 'C'
        SE = 'B'
        NE = 'A'
 )
```

Since the "DISPLAY AS SORTED" clause is not used, the region value of 'A' is considered to be greater than 'D' since 'A' is listed after 'D'. In a SELECT statement, a reference to REGION > 'C' would select region codes of 'B' and 'A'. A SELECT statement such as "SELECT IF REGION > 'A'" would result in an error message since 'A' is considered to be the highest (last listed) value of region.

Observation variables can be tested for null values, but if a null-valued variable is tested for a value other than null, the test will fail. For example, the test: x > 0 will fail if the value of x is NULL.

- **Arithmetic expressions**

Conditions can include arithmetic expressions. Any arithmetic expression allowed in the COMPUTE statement can be used in the SELECT statement. See the chapter called "Compute" for complete details on arithmetic expressions.

Arithmetic operations can contain only **observation** variables and numbers. The result of a computation can be compared to an observation variable, a number, or another arithmetic expression.

Example SELECT IF ANNUAL_INCOME / 12 > 2000;

- **Comparing one variable to another**

Format

```
name1 re name2
```

where the names refer to codebook or computed variables. **Name1** and **name2** must be the same type of variable: **observation**, **control** or **char**. For example, a control variable cannot be compared to an observation variable.

If **control** or **char** variables are being compared, their values must have the same justification and padding for the comparisons to work correctly.

Example

```
SELECT IF STATE_OF_RESIDENCE = STATE_OF_EMPLOYMENT;
```

- **Comparing a control variable to a condition name**

Format

```
control-name re control-condition-name
```

A **control** variable name may be used to identify one of its conditions by referencing it by condition name rather than condition value.

Example

```
(codebook)
STATE CON 2
  (
        ALABAMA = 1
        .   . .
        .   . .
        MARYLAND = 26
        .   .  .
  )

(request)
SELECT IF STATE = MARYLAND;
```

Sets of Values

Another type of condition lets you select records that have any of the values specified in **a set of values**. This feature is particularly useful if you need to select for a long list of values.

The format for entering a set of values in a SELECT statement is:

Format

```
SELECT IF  var  IN (val1, val2, val3, .... );
```

where **val1**, **val2**, **val3**, etc. are distinct values. Values must be separated by commas. Non-numeric values must be surrounded by quote marks. For observation variables, negative values such as -25 cannot be used in sets.

Comparisons such as less than or greater than cannot be used, but you can enter ranges of values separated by :, - or the word TO. For example, ranges such as 3:5, 3-5 or 3 TO 5 are allowed.

Example

```
SELECT IF INDUSTRY IN (1000, 2000, 3000:3999);
```

If the variable INDUSTRY has the value 1000, 2000, or any of the values 3000 through 3999, the record will be selected. This example gives the same result as the statement:

```
SELECT  IF INDUSTRY = 1000 OR INDUSTRY = 2000
          OR INDUSTRY >= 3000 AND INDUSTRY <= 3999;
```

The word IN cannot be preceded by the word NOT. If you want to select records that do NOT have any of the values in the set, you can specify:

```
SELECT IF  NOT  ( var IN (val1, val2, val3, .... ) );
```

Selecting records on the basis of a set of values will give more efficient processing than the individual comparisons if there are more than a few (e.g. 3 or 4) values in the set.

The clause **var IN (set of values)** can be used alone in the SELECT statement as shown in the preceding example, or it can be combined with other conditions. An example that combines the set of values with other conditions is:

```
SELECT IF REGION = 3 AND (INDUSTRY IN (410, 420, 425)
        OR INDUSTRY >= 450);
```

Records will be selected if they have REGION code 3 and INDUSTRY code greater than or equal to 450 or equal to 410, 420 or 425.

- **Sets of CHAR values**

If you are listing values for a CHAR variable and the values are all numeric, you do not need to enclose them in quotes. Non-numeric values must be enclosed in quotes. ***If they are shorter than the field width, you must enclose them in quotes and include any leading blanks or zeros.*** Otherwise, you will not get a match and nothing will be selected for these values. Trailing blanks need not be included in the quotes, but a good general rule is to make the value within the quotes have the same length and padding as the value in the data. The length should be the length given for the variable in the codebook. For example, if you have

```
INDUSTRY  CHAR  5
```

in the codebook, where some INDUSTRY values are less then 5 characters and filled on the left with blanks, an example of a correct SELECT statement for a set of INDUSTRY values is:

```
SELECT IF INDUSTRY IN ( '    410', '  4420', '66425' );
```

Compound Conditions

You can use compound conditions consisting of clauses separated by AND and OR. There is no limit to the number of compound conditions per statement. Conditions may be grouped by the use of parentheses to determine the order of evaluation. When parentheses are used, evaluation begins with the conditions contained in the inner-most sets of parentheses.

If the order of evaluation is not specified by parentheses, the expression is evaluated in the following order:

- arithmetic expressions
- relational operators
- AND and its surrounding conditions, starting at
  the left of the expression and proceeding to the right
- OR and its surrounding conditions, also
  proceeding from left to right

An expression such as:

```
SELECT IF A = B AND D > E OR F <=  WEIGHT * INCOME;
```

would be evaluated as:

```
SELECT IF ((A = B) AND (D > E)) OR
          (F <=  (WEIGHT * INCOME));
```

Using Record Names and COUNT in Conditions

Both record name variables and the built-in variable COUNT are observations variables that can be used in SELECT conditions wherever other observation variables are allowed. This means that you can select specific records or groups of records based on the record numbers.

In a file with only one record type, the record name variable and the built-in variable COUNT both take on the record number, so you can select records on either one with the same results. *Note* that in a hierarchical file, there can be more than

one type of record, so the record number is different from COUNT. See the chapter on "Hierarchies" for details.

Assume that in a file with only one type of record, the record name in the codebook is PERSON. We can select the 345th person record with the following SELECT statement:

```
SELECT IF PERSON = 345;
```

We can select the first 10 records with the statement:

```
SELECT IF PERSON <= 10;
```

In the following report request, we select a set of record numbers: 11, 89, 113, and 162.

```
SELECT if count in (11, 89, 113, 162);

RECODE hh_educ 'Education' on education;
label if all;

REPORT IN_SET 'Household records 11, 89, 113 and 162.':
count then state_code then hh_educ then income;
```

Household records 11, 89, 113 and 162.

Row	Count	State code	Education	Income
1	11	05	High School Graduate	37,535
2	89	27	Some College	30,750
3	113	15	High School Graduate	32,125
4	162	45	High School Graduate	20,316

Selection Using the NUMBER and PERCENT Options

The SELECT percent and SELECT number options of the SELECT statement let you select a subset of your data without regard to the data values. Instead, they allow you to select a specific section of your data or a randomly selected percentage of your data. These options are especially useful when you have a very large data file, because they enable you to experiment with your report requests without processing all of the data.

Format

```
SELECT  number % ;
        number PERCENT ;
        number ;
        number START number;
        number record-name ;
        number record-name START number;
```

Examples

```
SELECT  10 %;
SELECT  20;
SELECT  20 START 200;
SELECT  20 MEMBERS;
```

SELECT Percent

The SELECT percent statement gives you a representative sample of your data. If you select 1% of your data then as each record is read, a function is applied which gives it a 1% probability of being selected. *Note that the exact records selected and even the exact number of records selected is not fixed.* In some cases, more than 1% of the records will be selected. In other cases, less than 1% will be selected. If you run the same job multiple times, you will get different numbers in your reports each time.

Tip

Since the built-in variable COUNT has the record number as the value, you can use it in your report to see which records were selected. In the following example, we select approximately 2% of the records from a file containing 1052 records. In the run shown below, we get a report containing 19 records. We have included the variable COUNT in the report, so we can see that we have selected records 11, 89, 113, 162, 164 and so on.

```
SELECT 2%;

RECODE hh_educ 'Education' ON education;
LABEL IF ALL;

REPORT SEL_PCT 'Approximately 2% of household records':
COUNT THEN state_code THEN hh_educ THEN income;
```

Approximately 2% of household records.

Row	Count	State code	Education	Income
1	11	05	High School Graduate	37,535
2	89	27	Some College	30,750
3	113	15	High School Graduate	32,125
4	162	45	High School Graduate	20,316
5	164	16	8 years or less	13,901
6	268	02	College Graduate	36,263
7	338	02	Some College	4,666
8	450	25	Some College	73,550
9	495	25	High School Graduate	22,547
10	532	25	8 years or less	9,102
11	641	42	College Graduate	78,850
12	665	09	8 years or less	23,367
13	675	09	8 years or less	4,437
14	701	02	8 years or less	4,944
15	720	02	8 years or less	7,255
16	831	09	Post Graduate	67,500
17	853	09	High School Graduate	56,342
18	974	02	College Graduate	26,328
19	983	02	Post Graduate	6,016

If your data file is hierarchical, selection is done at the highest level of the hierarchy. For example, if you have a hierarchical file of families and family members, then either a family and all of its members are selected or the family and its members are not selected at all.

SELECT Number

The SELECT number option takes the first records from the file. If you select 10 records, it will be the **first 10** records.

If your file is a hierarchical file, for example, with family and member records, then the first 10 families and all of their members will be selected. If your SELECT statement is SELECT 10 MEMBERS; then exactly 10 members plus their family records will be selected. In this case, if the 10th member record in the file occurs before the last member record for its family, the members of the family that follow the 10th member record will not be selected.

The SELECT statement SELECT 10 START 5; will result in records 5 through 14 being selected.

Interaction Between Multiple SELECT Statements

The SELECT number and SELECT number START number statements are affected by other SELECT statements in the request. If there is a SELECT statement at a higher hierarchical level or if a SELECT statement at the same level occurs earlier in the request, then the SELECT number statement applies to records which pass the earlier SELECT.

Examples

Suppose that your data file contains information about persons, where there is one record per person, and that your report request contains the statements:

```
SELECT IF SEX = 'm';
SELECT 10 START 5;
```

Further suppose that the first 20 records have a sex of female and the next 20 records have a sex of male. TPL REPORT will exclude the first 20 records because they fail to pass the first SELECT. It will then skip records 21 to 25 because of the START clause. It will include records 26 through 35 and exclude the remainder of the file.

If the SELECT statements are reversed, records 1 through 5 will be excluded by the START clause. Records 6 through 15 will pass the SELECT 10 clause but will fail the SELECT IF SEX = 'm' condition. Records 16 to the end of the file will be excluded by the SELECT 10 clause. Thus no records will pass the two SELECT statements.

CHAPTER 10

Compute

COMPUTING NEW VARIABLES

Format

```
COMPUTE new-variable ['print label'] [USING MASK mask] = computation;
                                     USING          :
                                     MASK
```

where **new-variable** is an **observation** variable.

The optional **print label** following the variable name replaces the variable name on the printed report. There are many options associated with print labels such as upper and lower case letters, special characters and alignment. A separate chapter describes print labels in more detail.

The optional **mask** is used to specify the print format for the computed data values if the computed variable is used in a REPORT statement.

Examples

```
COMPUTE MONTHLY_INCOME = ANNUAL_INCOME / 12;

COMPUTE NO_FAMS 'Number of Families'
        USING MASK 9999 = FAMILIES;

COMPUTE FACTOR = .571

COMPUTE WEIGHTED_INCOME USING MASK '$'999,999 =
        INCOME * WEIGHT;

COMPUTE RATIO  MASK 9.999 = LOAN_AMT / PROPERTY VALUE;
```

Introduction

The COMPUTE statement provides a way of creating a new observation variable which has not been defined in the codebook. The new variable is calculated from other observation variables (including a record name) and numeric literals using addition (+), subtraction (-), multiplication (*), division (/) and exponentiation (**).

The calculations are performed using the normal rules for evaluation order and parentheses. Unless parentheses are used to change the evaluation order, exponentiation is performed before division and multiplication, which, in turn, are performed before addition and subtraction. Strings of operations at the same level are performed left to right. For example, 10 - 5 + 6 is evaluated as (10 - 5) + 6 = 11, not as 10 - (5 + 6) = -1.

A special division operator called **DIV** is available for performing integer division, and the **SQRT** and **ABS** functions can be used to get square roots and absolute values.

A computed variable is always considered to be observation variable and can be used in following TPL statements in any place that a codebook-defined observation variable can be used.

COMPUTE statements are executed in ANSI standard double precision floating point.

Compute Entries

A computation can reference numeric literals and observation variables from either the codebook or previous COMPUTE or Conditional Compute statements. The numeric literals can contain an actual decimal point. Parentheses can be used in the computation to any level. Computed values are rounded, if necessary, just before being displayed. An optional mask to the left of the equal sign indicates the number of decimal places and special symbols to be displayed.

The statement:

```
COMPUTE AMT 'Expenditure Amount' =  ((INTEREST + 3)  /
        (GROSS - INTEREST)) * .5;
```

is a valid COMPUTE statement if INTEREST and GROSS are observation variables from either the codebook or a previous COMPUTE or Conditional Compute statement. A computed variable can be set equal to a constant value or to another observation variable.

For example:

```
COMPUTE A = 5;
COMPUTE B = + 3;
COMPUTE C =  -95;
COMPUTE FAMILY_INCOME = INCOME;
```

The "new-variable" being computed cannot have the same name as any other variable created or used within the report request. If it has the same name as a variable in the codebook named by the USE statement, references to that name will be assumed to refer to the computed variable rather than the codebook variable.

For example:

```
COMPUTE WAGES = INCOME  / 50;
```

If a variable called WAGES has already been created in an earlier statement, an error will be reported.

If **division by zero** is attempted, a warning message will be issued and a **(c)** will be reported.

If any variable used in a computation has a **NULL** value, then the computed variable will be assigned a NULL value. For example, if you specify DATA ERROR = NULL when describing the variable INCOME in the codebook, and you then use INCOME in a COMPUTE statement, the computed result will be NULL for any record that has a data error for the variable INCOME. In a report, a NULL value is displayed as **(n)**.

Absolute Value

You can obtain the absolute value of any expression within a computation by enclosing the expression in parentheses and preceding the left parenthesis with the keyword **ABS**, as in:

```
COMPUTE AMT 'Weighted Income' = ABS (WEIGHT  * INCOME);
```

Square Root

Square root can be obtained by following the keyword **SQRT** with a computation within parentheses. If square root is attempted on a negative value, a warning message will be issued and a value of zero will be returned.

Integer Division

If you want division to be performed in integer mode so that all decimal places are truncated for each computed result, then use the **DIV** operator in place of the divide symbol (/). For example, 3 DIV 4 is zero, whereas 3/4 is 0.75. A DIV by 1 will simply truncate the decimal places. For example, 2.688 DIV 1 gives a result of 2.

Note

The DIV function can only guarantee 11 to 12 digits of accuracy. The reason for this is that TPL has code which prevents possible larger errors when data values are converted to floating point in the computer. Suppose the data value is really 7.000000000000000 but because of data conversion errors the number is represented in the computer as 6.999999999999999. If we do a straight truncation for DIV we will get an incorrect number, 6, even at the integer level. TPL corrects for this by adding a small number to the value before truncation. Hence the value becomes something like 7.000000000000754. Now if we truncate to integers or even to 7.0000000, we get correct numbers but if we truncate to 7.0000000000007 we get an incorrect value. TPL opts to give up a few digits of accuracy to prevent errors from showing up in higher digits.

Masks for Output Formatting

The COMPUTE statement can also contain a concise expression of how the computed variable is to be formatted when printed. This expression, known as a **mask**, consists of a succession of **9's**, one for each digit position of the expected maximum aggregated value.

Commas and a decimal point can be embedded within the 9's. For example, a mask of **9,999.99** would cause a data value of 2467.34 to be displayed as **2,467.34**. Without the mask it would be displayed as 2,467.

Alphabetic or other special characters can be placed at the beginning or end of a mask if they are enclosed in quotes.

Example

```
COMPUTE FAMILY_INCOME USING MASK '$'999,999.99 =
        (HEAD_INC + OTHER_INC) / 100;
```

The total of HEAD_INC and OTHER_INC, which are recorded in cents, are to be added together for each family and displayed in dollars and cents. A dollar sign, comma, and decimal point are also to be printed. Since the incomes are to be expressed in dollars and cents, the totals must be divided by 100 to move the decimal point two positions to the left. Note that if HEAD_INC and OTHER_INC were

desribed in the codebook with the clause SHIFT DECIMAL LEFT 2, we would not need to divide by 100 in order to display the incomes in dollars and cents.

Data can be rounded and displayed with trailing zeros by inserting zeros in the mask. For example, with a mask of 999,000 the value 876859 will be displayed as 877,000.

For additional details on the use of masks, see the chapter on Masks.

Weighting

A common need in statistical processing is to weight various observed values in a data record which represent a sampling. Typically, each processing unit contains a weighting factor to be applied to the entries, so that the final values represent a larger universe.

Weights are observation variables that can be displayed in reports the same as any other observation variables. You can also use weights in COMPUTE statements to create weighted values in reports. Computations of this type would be especially useful if you were using TPL REPORT to create a data file in which the weight variable was already applied to the observations variables. The FORMAT statement called DATA REPORTS can be used to create a data file from a report.

A weighted variable can be created by multiplying the variable to be weighted by the weight factor. For example,

```
COMPUTE WEIGHTED_INCOME = INCOME * WEIGHT_FACTOR;

COMPUTE WEIGHTED_EXPENSE = EXPENSE * WEIGHT_FACTOR;
```

where, INCOME, EXPENSE, and WEIGHT_FACTOR are codebook observation variables. WEIGHTED_INCOME and WEIGHTED_EXPENSE are computed for each record. They can then be used in a REPORT statement such as:

```
REPORT T1:  AUTO_TYPE  REGION  WEIGHTED_INCOME
        WEIGHTED_EXPENSE;
```

The computed weighted values will be reported for each record.

THE CONDITIONAL COMPUTE STATEMENT

Introduction

An extension of the COMPUTE statement can be used to create an observation variable for which the computation varies depending on the values of one or more other variables. The new variable will be assigned the computation associated with the first condition satisfied. This type of COMPUTE statement is called Conditional Compute.

The Statement

Format

```
COMPUTE new-obs-var ['print label'] [USING MASK mask] =
                                    USING
                                    MASK

computation-1  IF condition-1A [AND condition-1B...];
NULL           :               OR

[computation-2 IF condition-2A [AND condition-2B...]; ]
NULL           :               OR
       .            .   .
       .            .   .
[computation-n  IF  OTHER; ]
NULL            :
```

The keyword 'IF' and the colon can be used interchangeably.

Examples

```
COMPUTE WEIGHTED_INCOME  =
        WEIGHT * INCOME    IF  INCOME > 20000;
        INCOME             IF  OTHER;

COMPUTE TEST_ZERO_DENOMINATOR  =
        EARNINGS / HOURS   IF  HOURS > 0;
        NULL               IF  OTHER;

COMPUTE AMT3 USING MASK 999 =
        A        IF  L > 25.55;
        L        IF  OTHER;
```

```
COMPUTE CHILDREN USING MASK 99 =
        PERSONS_IN_FAMILY - 2  IF
                MARITAL_STATUS = 1 AND SEX = MALE;
        PERSONS_IN_FAMILY - 1  IF
                MARITAL_STATUS  IN  (1, 2, 3, 5);
                0  IF  OTHER;           /* Never Married and */
                                        /* Not available */
```

Condition Term

The variables used to the right of the 'IF' in the Conditional Compute can be either **control** variables (but not a recoded variable), **char** variables or **observation** variables.

The conditions are expressed identically to the IF form of the SELECT statement. Conditions can test for **relationships** or **sets of values**. All conditions which are valid for the SELECT statement are valid for the Conditional Compute. Each condition can reference entirely different variables.

Compute Term

The entries to the left of the 'IF' can be:

1. numeric literals and observation variables from the codebook or computed variables. A record name cannot be used.

or,

2. the keyword NULL as explained later in this chapter.

As a first example, suppose that the control variable PAY_TYPE contains either an 'H' or a 'W' to indicate whether the observation variable EARNINGS is stored as an hourly wage rate in cents or a weekly salary in dollars. To create a new observation variable WEEKLY_SALARY to be displayed in dollars and cents, we could write:

```
COMPUTE  WEEKLY_SALARY USING MASK  '$' 999.99 =
        (EARNINGS * USUAL_WEEKLY_HRS)/100       IF  PAY_TYPE = 'H';
        EARNINGS/100                            IF  PAY_TYPE = 'W';
```

Note that if EARNINGS had been described in the codebook with the clause SHIFT DECIMAL LEFT 2, we would not need to divide by 100 in order to display the results in dollars and cents.

The ordering of the entries is important in the Conditional Compute. The conditions are evaluated in the order in which they are specified. ***The new variable will be assigned the computation associated with the first condition satisfied.***

Consider an expansion of the last example:

```
COMPUTE WEEKLY_SALARY USING MASK '$' 999.99 =
        (EARNINGS * USUAL_WEEKLY_HRS)/100      IF  PAY_TYPE = 'H';
        EARNINGS/100                           IF  PAY_TYPE = 'W';
        EARNINGS                               IF  PAY_TYPE = 'H';
```

If PAY_TYPE equals 'H', WEEKLY_SALARY will be set equal to the first computation and no more testing will be done. Although 'H' occurs more than once, any computation other than the one associated with the first occurrence of 'H' will be ignored.

If none of the conditions are satisfied, then the new variable will be assigned the value of zero. Thus in the above example, if PAY_TYPE='H', then the new variable WEEKLY_SALARY will contain the value of the computation expressed by (EARNINGS * USUAL_WEEKLY_HRS)/100. If PAY_TYPE='W', then WEEKLY_SALARY will contain the value EARNINGS/100. If neither condition is satisfied, WEEKLY_SALARY will be assigned the value zero.

If PAY_TYPE cannot take values other than 'H' AND 'W', the statement could have been written using 'OTHER' as in,

```
COMPUTE  WEEKLY_SALARY USING MASK '$' 999.99 =
        (EARNINGS * USUAL_WEEKLY_HRS)/100      IF  PAY_TYPE = 'H';
        EARNINGS/100                           IF  OTHER;
```

In the Conditional Compute statement at least one computation or numeric literal must appear in the first entry in the column under the new variable. If subsequent entries do not contain a computation on the left, they will be associated with the previous computation. For example, if we need a new variable **WEIGHT** which varies depending on the value of the variable STATE, we could use the following statement. Codebook condition names associated with STATE are used.

```
(codebook)
STATE CON 2
  (
        ALABAMA = 1
                . . .
                . . .
         WYOMING = 50
  )
```

```
(request)
COMPUTE WEIGHT =
        1       IF  STATE = Arizona;
                IF  STATE = Utah;
                IF  STATE = Nevada or STATE = New_Mexico;
        3       IF  STATE = California;
                IF  STATE = New_York;
        2       IF  OTHER;
```

WEIGHT will have a value of 1 if STATE = Arizona, Utah, Nevada or New Mexico, and a value of 3 if STATE = California or New York. All other state values will cause weight to have a value of 2.

The same computation can be repeated in different entries. For example, to create WEIGHT we might wish to list the states in alphabetic order on the right as follows:

```
COMPUTE WEIGHT =
        2       IF  STATE = Alabama;
        2       IF  STATE = Alaska;
        1       IF  STATE = Arizona;
        2       IF  STATE = Arkansas;
        3       IF  STATE = California;
        2       IF  STATE = Colorado;
        2       IF  OTHER;
```

An observation variable can be tested and also used in the computation. For example:

```
COMPUTE AVG_PRICE =
        (PRICE1 + PRICE2 + PRICE3) / 3
                IF    PRICE1 > 0 AND PRICE2 > 0 AND PRICE3 > 0;
        0       IF    OTHER;
```

A conditionally computed variable can be used anywhere that an observation variable can appear, including in another Conditional Compute statement.

If the computation associated with the first condition satisfied contains a null value, the new variable value for that record will be null-valued.

NULL Values

A null value can be assigned to a conditionally computed variable by using the keyword NULL on the left side of a condition. NULL values are displayed in reports as **(n)**.

Suppose that we want a report showing employees and their hourly salaries but, for employees with a salary above $20 per hour, we want to suppress the salary value in the report. We can do this with the following conditional compute:

```
COMPUTE SALARIES MASK 99.99 =
HOURLY_WAGE        IF      HOURLY_WAGE <= 20;
NULL               IF      OTHER;
```

In SELECT and Conditional Compute statements, no test involving a null-valued variable will succeed unless it specifically references NULL. For example, if variable "A" has a null value, then "A = 5" will not be satisfied. Similarly, "A NOT = 5" will not be satisfied. If all of the variables referenced in a Conditional Compute are null-valued, all of the tests will fail and the newly computed variable will take on the value associated with the "OTHER" category if one is provided; it will take on the default value of zero if no "OTHER" category is specified.

The keyword NULL can be explicitly used in the SELECT statement and on the right side of a Conditional Compute or a RECODE statement. For example, suppose that there are two income variables called MONTH_EARNINGS and WEEK_EARNINGS. Suppose also that one of the two incomes can be null-valued. We want to compute a new variable EARNINGS which is expressed in monthly earnings. We could use a statement like the following:

```
COMPUTE EARNINGS =
  MONTH_EARNINGS  IF  MONTH_EARNINGS NOT = NULL
        AND  WEEK_EARNINGS = NULL;
  (52 / 12) * WEEK_EARNINGS IF  WEEK_EARNINGS NOT =  NULL
        AND  MONTH_EARNINGS = NULL;
  NULL IF  OTHER;
```

In a RECODE statement, null values are not included in any RECODE category, including OTHER, unless they are specifically mentioned. If null values are present for the old variable and not mentioned in the RECODE, the recoded value will display as blank for any null value.

CHAPTER 11

Recode

REPLACING ORIGINAL VALUES WITH LABELS OR NEW VALUES

Introduction

With the RECODE statement, you can create a new variable based on the values of an existing observation or control variable and assign labels or combinations of labels and values in place of the values that would be printed by default for the original variable. You can also report the same information for a group of values.

Variables created with RECODE can be used only in REPORT statements.

Format

```
RECODE new-variable-name ['variable label']   ON  old-variable-name;
                                                  :

    [new-category-1]     IF     [re] value-entry-1;
                                :
    [new-category-2]     IF     [re] value-entry-2;
                                :
          .                            .
          .                            .
          .                            .
    [new-category-n]     IF     [re] value-entry-n;
                                :
```

A **new-category** entry can be any of the following:

- label
- name
- blank entry
- LABEL (if old variable is CONTROL)
- NAME (if old variable is CONTROL)
- VALUE
- VALUE(n) (if old variable is OBS)
- VALUE(-n) (if old variable is OBS)

A **value-entry** can be any of the following:

- value
- condition name (if old variable is CONTROL)
- value1 : value2
- OTHER
- ALL
- NULL

If a **value-entry** does not have a name or label assigned to it, it is grouped into the first category above it that does have a name or label.

The optional **re** stands for any relation symbol or the equivalent English as shown below. A relation symbol can precede any value. If no relation is provided, "equal" is assumed.

Symbol	English expression
<	[IS] LESS THAN
>	[IS] GREATER THAN
=	[IS] EQUAL [TO] EQUALS
^<	[IS] NOT LESS THAN
^>	[IS] NOT GREATER THAN
^=	[IS] NOT EQUAL [TO]
<=	[IS] LESS THAN OR EQUAL [TO]
>=	[IS] GREATER THAN OR EQUAL [TO]

Example

Assume that we have data on employees and that an employee's hourly salary is contained in a variable called RATE. If we were to use RATE in a REPORT statement, the hourly salary rate would be reported for each record.

If we wish to report salary rates in categories, we can use a RECODE to create salary categories based on the original hourly rates:

```
RECODE  RATE_CATEGORIES 'Hourly rates'  on  RATE;
'Under $6.00'          IF      < 6.00;
'$6.00 to 9.99'        IF      6.00 : 9.999;
'$10.00 and over'      IF      OTHER;
```

If we then use both RATE and RATE_CATEGORIES in a REPORT statement, we can see how the RECODE works.

```
REPORT RC1  'Rates and Rate Categories' :
        RATE THEN RATE_CATEGORIES;
```

Rates and Rate Categories

Row	Pay Rate	Hourly rates
1	5.53	Under $6.00
2	21.49	$10.00 and over
3	7.35	$6.00 to 9.99
4	9.21	$6.00 to 9.99
5	5.00	Under $6.00
6	5.19	Under $6.00
7	8.65	$6.00 to 9.99

Description of the RECODE Statement

The RECODE statement is in the form of a two column table. When we refer to rows in the description below, we really mean just a left/right pair of entries. Except for the order of entries the statement is free format and you can group your entries in whatever way is convenient.

The first row starts with the keyword RECODE followed by the name of the new variable to be created and, optionally, a label for the new variable. This is followed by either 'ON' or ':', then an old variable name. In following rows, entries for the two columns are separated by either 'IF' or ':'. Each line must end with a semicolon.

When the new variable is subsequently used in a report, its name will be displayed at the top of its report column unless you provide a variable label. In that case,

the variable label will be displayed instead of the name. There are many options associated with print labels such as upper and lower case letters, special characters and footnotes. A separate chapter describes print labels in more detail.

The new variable created by the RECODE can be used in any REPORT statement but not in other types of statements. The old variable can be either control or observation and can be either a codebook variable or a computed variable. *It cannot be a CHAR variable*.

After the RECODE row, one or more additional rows of recode information follow. Each row contains a left/right pair of entries in which the entry on the left is assigned to the recode condition on the right. If there are multiple rows with the same condition on the right, only the first row will be used for that condition.

New Variable Entries on the Left

The left side specifies what should be displayed for each RECODE category. *For any values of the old variable that are not included in the RECODE statement, the original values will be used as the recode values.*

An entry on the left side can be:

1. A **label**. Any valid TPL REPORT label is allowed. **Note** that you can also use this type of entry to enter recode values that are numeric, but you must enclose them in quotes. An example is '1'.

2. A **name**. Any valid TPL REPORT name is allowed. The name is used as a simple label. Any alphabetic characters in the name are converted to upper case before they are displayed in the report.

3. A **blank** entry. If any left-side entry is not filled in (blank entry), it will take its assignment from the nearest row above that has an explicit entry in the left. This has the effect of grouping the values of the old variable into the same recode category.

4. The word **LABEL**, if the old variable is a CONTROL variable. The original condition label from the codebook is assigned to the recode category. LABEL can only be used alone. In other words, it cannot be combined with other label elements.

5. The word **NAME**, if the old variable is a CONTROL variable. NAME can be used alone or with other label elements. The original condition name from the codebook is assigned to the recode category. If there is no condition name for the value, nothing will be displayed where NAME is used. In this case, if NAME is used alone, without other label elements, nothing will be displayed in the report for that recode category.

6. The word **VALUE**. It can be used **alone or with other label elements** to assign the original old variable value to the recode category;

   If the old variable is an OBSERVATION variable, you have the option of including a number in parentheses, **VALUE(n)**, to display decimal places.

   In the following example, the original values for pay rate are shown alongside the recoded values. The RECODE replaces each original value with a combination of label elements and the value displayed to 2 decimal places.

```
RECODE REC_RATE   'Recoded Rate' ON RATE;
RIGHT FONT TIU 'Rate ' FONT HB VALUE(2) IF ALL;

REPORT RC2 'Pay Rates and Recoded Rates':  RATE  REC_RATE;
```

Rates and Recoded Rates

Row	Pay Rate	Recoded Rate
1	5.53	*Rate* **5.53**
2	21.49	*Rate* **21.49**
3	7.35	*Rate* **7.35**
4	9.21	*Rate* **9.21**
5	5.00	*Rate* **5.00**
6	5.19	*Rate* **5.19**
7	8.65	*Rate* **8.65**

   If you include a negative number, **VALUE(-n)**, the value will be rounded, for example to hundreds or thousands, with the appropriate number of zeros displayed at the end of the value. For example, use of VALUE(-2) will round each value to the nearest hundred and display it with two zeros at the end.

NAME and VALUE cannot be used for the same recode category. For a single category, there can be only one use of either one.

Old Variable Entries on the Right

The right side contains the values of the old variable that will be recoded in the new variable. A semicolon is required at the end of each entry.

Valid entries are:

1. An old variable **value**, for example **1000** or **'F'**.

   If the old variable is a **control** variable, condition values that are not all numeric must be surrounded by quote marks according to the rules for listing values in the codebook.

2. A **condition name** from the codebook if the old variable is a CONTROL variable. An example is **MALE.**

3. A **relation followed by a value or name**, where relation is any of those listed earlier in this chapter. An example is **< 25**.

4. A **range** of old variable values specified as **value1 : value2** where value1 is less than or equal to value2. For example, if **2:5** is specified, all values not less than 2.000... and no greater than 5.000... will be accepted. A range of values can be non-numeric, in which case each lower and each upper value must be surrounded by quote marks; an example is **'A':'D'**. The keyword **TO** can be used in place of **:** to separate lower and upper range values. An example is **2 TO 5**.

   The NOT operator can precede a range of values, e.g. NOT 2:5, means any value that is either less than 2.000... or greater than 5.000....

   The relations > and < can also be used in ranges in certain circumstances. The valid formats are listed below. The value **m** must always be less than the value **n**. If the old variable is an observation, the values can contain decimal points.

   ```
   [NOT]        m : n;
              > m : n;
                m : < n ;
              > m : < n ;
   ```

5. The word **OTHER**, meaning all values for the old variable which are not specified by any other entry. If OTHER is used, it must be the last entry in the RECODE statement.

   You can use OTHER as the only entry in the statement in the same context as that described below for ALL.

6. The word **ALL** if you wish to apply the same treatment to all values. If ALL is used, there can be only one entry in the RECODE statement. For example, to display the condition labels of a control variable such as REGION rather than displaying the values, we could create a new variable with only the labels:

   ```
   RECODE REGION_WITH_LABELS  ON  REGION;
   LABEL  IF  ALL;
   ```

7. The word **NULL** to reference null values for observation variables. Assume, for example, that we have a variable called INCOME described in the codebook with the clause DATA ERROR = NULL . By default, null values will be reported as the letter **n** in parentheses **(n)**. If we wished to report the null values with a label instead, we could do it with the following statement:

   ```
   RECODE NEW_INCOME  ON  INCOME;
   'Missing value'  IF  NULL;
   ```

   Null values are not included in any RECODE category, including ALL and OTHER, unless they are specifically mentioned. If null values are present for the old variable and not mentioned in the RECODE, the recoded value will display as *blank* for any null value.

Unspecified Values

If there are any values of the old variable that are not specified, these values will be displayed in the report as if the RECODE statement ended with the entry:

```
VALUE  IF  OTHER;
```

Note on Value Order in Relations and Ranges

If the old variable is a control variable and has been described in the codebook using the default order DISPLAY AS LISTED, then all relations with values or ranges of values must be based on the order of the conditions listed, rather than the sort sequence of the condition values. For example, consider the following control variable entry:

```
REGION CON 1
  (
        SW      =  'D'
        NW      =  'C'
        SE      =  'B'
        NE      =  'A'
  )
```

Unless the DISPLAY AS SORTED clause is used, the region value of 'A' is considered to be greater than 'D' since 'A' is listed after 'D'. A RECODE statement used to combine "SE" and "NE" into one classification would have to express the condition as 'B':'A' or as > 'C'. Likewise, a recode condition such as > 'A' would have no matching values in the data, because 'A' is the last and therefore the highest value for region.

More RECODE Examples and Applications

Grouping Values into Larger Categories

Assume that we have a variable called STATE with values from 1 to 50. Instead of displaying the individual state values in our report, we want to print the name of the region to which each state belongs.

If the state values are grouped by region, we can recode them using simple ranges. For example:

```
RECODE REGION 'Region' ON STATE;
'Northeast'     IF      1 TO 9;
'Midwest'       IF      10 TO 22;
'South'         IF      23 TO 40;
'West'          IF      41 TO 50;
```

Alternatively, if the states are coded in alphabetical order of state name, the state values will not be neatly grouped by region. In this case, we can recode them using the individual values, mixing these entries with ranges whenever possible. For example:

```
RECODE REGION 'Region' ON STATE;
'Northeast'             IF      25;
                        IF      11;
                        IF      42;
                        IF      30 : 35;
'Midwest'               IF      12 : 14;
                        IF      5;
                        IF      8;
etc.....
```

Suppressing Display of Selected Values

Assume that we are preparing a report in which we wish to display annual INCOME values, but we do not want to show incomes greater than $200,000. We can replace these values with the word 'Confidential' while showing the actual incomes for all other values.

```
RECODE SELECTED_INCOMES 'Income'  ON  INCOME;
'Confidential'    IF       > 200000;
```

Note that for any INCOME values not mentioned in the RECODE statement, the actual values will be displayed. This is the same as the result we would get by adding the following line to the RECODE:

```
VALUE  IF  OTHER;
```

Replacing Values with their Labels

Assume that we have a CONTROL variable called INDUSTRY in the codebook and that there are hundreds of industry codes. The code values are are numeric, but we have included industry labels in the codebook for each value. By default, if we use INDUSTRY in a report, the industry code values will be displayed. Values such as 1220 or 1401 are not very informative, so we might prefer to use the industry code labels in the report. With the following RECODE, we can do this easily, without re-entering the industry code labels:

```
RECODE IND_CODE 'Industry'  ON  INDUSTRY;
LABEL  IF       OTHER;
```

Since no individual industry codes are referenced, OTHER will apply to all industry codes and LABEL will cause the codebook labels to be used in the report.

A Combination of Labels and Values

Assume that we have the following variable described in the codebook:

```
COLLEGE 'COLLEGE & UNIVERSITY TRAINING' CONTROL 1
(
        0:9
'NOT REPORTED'  = ' '
'NOT REPORTED'  = 'Y'
'AGENCY DELETED' = '*'
)
```

We want to display the codes for values 0 to 9, but for the other values, we want to display 'NA'. We can do this with the following RECODE:

```
RECODE COLL_UNIV 'Training'  ON  COLLEGE;
'NA'            IF      ' ';
                IF      'Y';
                IF      '*';
VALUE           IF      OTHER;
```

Creating a New Data File with Recoded Values

Assume that we want to extract a subset of the variables in a data file and create a new, smaller data file using the FORMAT statement DATA REPORT; as described in the FORMAT chapter. We may also want to group values of one or more old variables to create new categories for the new data file.

For example, if we have state codes in the original data file but we only want to work with a region code, we can create the region codes with RECODE and include only the new variable in the data file output. In the RECODE, we can assign numeric region codes by entering them in quotes.

```
RECODE REGION ON STATE;
'1'             IF      1 : 9;
                IF      12 : 14;
'2'             IF      10 : 11;
                IF      15 : 22;
'3'             IF      23 TO 40;
'4'             IF      OTHER;
```

Results with Overlapping Ranges

Assume that we have data on hourly SALARY rates. Instead of reporting the actual rates, we would like to categorize them as 'Under $5', '$5 to $10', etc. We can do this with the following RECODE:

```
RECODE SALARY_GROUPS 'Salaries' ON SALARY;
'Under $7'      IF      < 7.00;
'$7 to $12'     IF      <= 12.00;
'Above $12'     IF      OTHER;
```

Note that when a SALARY value fits into more than one RECODE category, it will be assigned to the first one that fits. In this example, the value 5.55 would fit into both the first and second categories since it is both less than 7 and less than 12, but it will be assigned only to the 'Under $7' category.

If we reversed the order of the first two rows, we would get a different result. Since all values that are less than 7 are also less than 12, all values less than 12 would go into the first category and none would go into the 'Under 7' category.

Recode on a Record Name Variable or COUNT

In a file with only one record type, the record name variable and the built-in variable COUNT both take on the record number, so a recode of any of these variables assigns new values depending on the record number. *Note* that in a hierarchical file, there can be more than one type of record, so the record number is different from COUNT. See the chapter on "Hierarchies" for details.

In the following example reporting data from a file with only one type of record, we assign the label value 'below #3' to records 1 and 2. The rest of the records get the label value '3 and over'. As the table illustrates, the results are the same for both the record name EMPLOYEE and the COUNT variable.

```
USE JOBS CODEBOOK;

RECODE ON_COUNT ON COUNT;
'below #3'              IF < 3;
'3 and over'            IF OTHER;

RECODE ON_RECORD ON EMPLOYEE;
'below #3'              IF < 3;
'3 and over'            IF OTHER;

REPORT REC1 'Report with recode of record name (EMPLOYEE)'
                'and COUNT.':
        EMPLOYEE  ON_RECORD  COUNT  ON_COUNT  RATE;
```

Report with recode of record name (EMPLOYEE) and COUNT.

Row	Employee	ON RECORD	Count	ON COUNT	Pay Rate
1	1	below #3	1	below #3	5.53
2	2	below #3	2	below #3	21.49
3	3	3 and over	3	3 and over	7.35
4	4	3 and over	4	3 and over	9.21
5	5	3 and over	5	3 and over	5.00
6	6	3 and over	6	3 and over	5.19
7	7	3 and over	7	3 and over	8.65

CHAPTER 12

Char

CREATING A NEW CHARACTER VARIABLE

The CHAR statement creates a new character variable by combining all or part of other character variables and text. Its primary use is in creating variables for use in TPL reports. It also may be used in TPL TABLES to create variables for use in SELECT statements.

Format

```
CHAR new-variable ['print-label'] = construction ;
```

where **new-variable** is a character variable and **construction** is made up of one or more of 'character-string', substr(character-variable, start) and substr(character-variable, start, length) concatenated together using '+' or '||'.

Example

In our first example we wish to select all **items** which begin with 'A';

```
CHAR SELECT_VAR = SUBSTR(ITEM,1,1);
SELECT IF SELECT_VAR = 'A';
```

In our next example we wish to create a TPL report which includes the full names of people in a data file. The data is stored with first name, FNAME, in one field and last name, LNAME, in another. If we just use FNAME and LNAME in the report, then the fields will be in separate columns with varying numbers of blanks between them. Instead we use:

```
CHAR FULL_NAME 'Name' = FNAME + ' ' + LNAME;
```

Note that we have included a blank between FNAME and LNAME since otherwise the fields would be run together.

CHAR SPLIT: DIVIDE A CHARACTER VARIABLE

The SUBSTR function can be used to split a fixed-format data field into parts. If the data is not fixed-format, then CHAR SPLIT must be used.

Format

CHAR SPLIT old-variable: $variable_1$ "$divider_1$" $variable_2$ "$divider_2$" ...
CHAR SPLIT old-variable: "$divider_1$" $variable_1$ "$divider_2$" $variable_2$ ...

where old-variable is the variable which is to be divided into two or more new variables. The dividers, which are a list of characters, are entered in quotes. They separate the subfields in the data.

Example

Suppose your CSV data file has a DATE field. The field on different records are:

```
1/23/45     and
12/22/46
```

You wish to split the DATE field into three new observation variables, MONTH, DAY, and YEAR. You can do this with:

```
CHAR SPLIT DATE: CMONTH "/" CDAY "/" CYEAR;
COMPUTE MONTH = OBS(CMONTH);
COMPUTE DAY = OBS(CDAY);
COMPUTE YEAR = OBS(CYEAR) + 1900;
```

Note

Dividers need not be the same from subfield to subfield. All characters in the divider list between two subfields are discarded. regardless of order.

Example

Suppose you have a data field **VALUES** which is: [+ 34 +A 23] and you wish to extract just the two numbers. You can use

```
CHAR SPLIT VALUES: " +" VALUE1 "A+ " VALUE2;
```

VALUE1 will get the value 34 while VALUE2 will get 23.

CHAPTER 13

Hierarchies

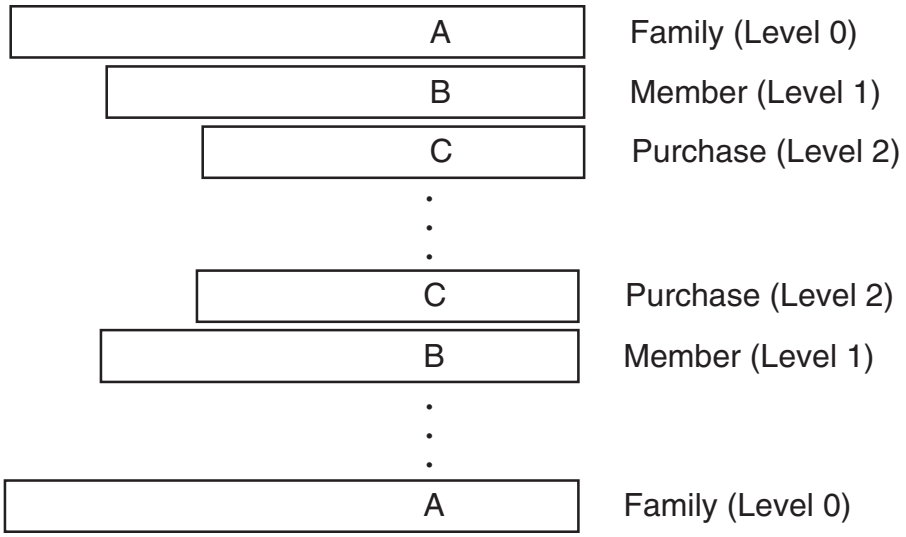
PROCESSING HIERARCHICAL FILES

Introduction

A hierarchical file consists of multiple record types, each related to the other but describing a different level of detail. The records are sequenced so that for records at any level of detail, a variable number of more detailed records may follow.

You can request reports from hierarchical files as will be described in this chapter. A repeating group is another type of data structure that has many of the attributes of a hierarchical file. Repeating groups are described in a separate chapter.

The following diagram illustrates the concept of hierarchically related records. Lower level records are shown indented to suggest subordination within the hierarchy. Records containing the greatest detail are dependent on the next higher level of the hierarchy and so on back up to the Master level. The master record has a level number of zero indicated by LEVEL 0. The record type immediately subordinate to the Master record is LEVEL 1. Level numbers increase in increments of 1 to the level number of the lowest order of subordination in the hierarchy. The structure of the three-level FAMILY hierarchical file is shown next, followed by the codebook description. Note that some unique value (e.g., A,B,C) within each record must uniquely identify each level of the hierarchy.



Each level of the hierarchy is identified in the codebook by a record marker and a level number. For example, each family record may be identified by a marker of **1** in record position 1 and each member record may be identified by a marker of **2** in record position 1.

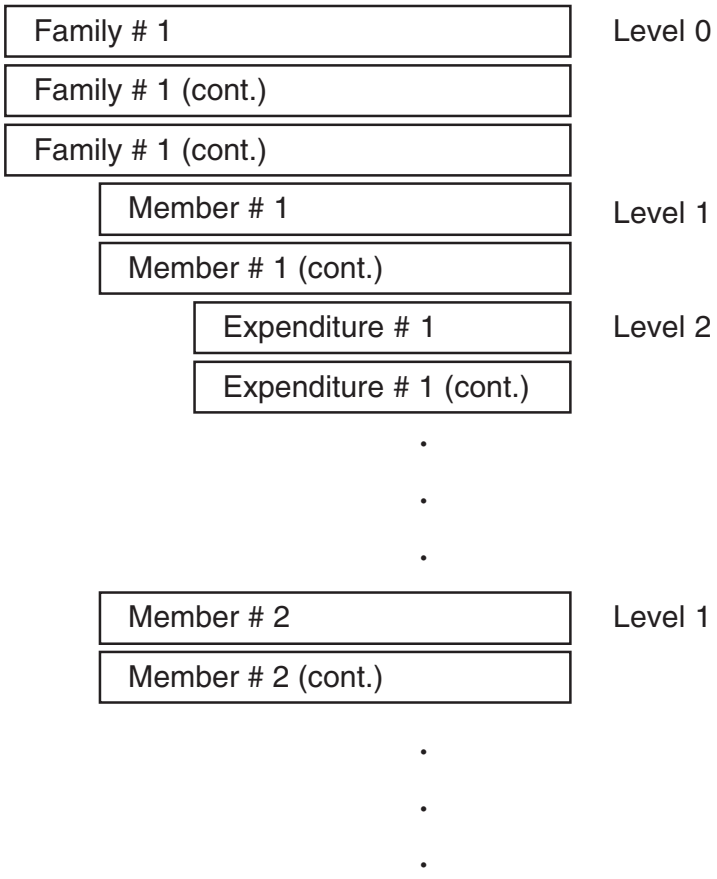
Let us assume that each family record in a data file has one or more family members, and each member record has at least one purchase record associated with it. When processing this file, TPL REPORT will read the first family record, the first member record, and the first purchase record. These three levels will form a hierarchical unit for reporting. Next, the second purchase record (if any) will replace the first to form another hierarchical unit.

After all purchase records have been combined individually with the first family and member record, the second member record is read, if any. The first family record is paired with the second member record and each purchase record of that member in turn. After all member records for the first family and their purchase records have been processed, the next family record is read and the cycle is repeated.

Each record or collection of records is assigned a level number whose value depends on its hierarchical relationship to other records. The record type which identifies a major new processing unit in the file, and which is not subordinate to other records, is known as a master record or level 0 type record. In our example, the family record is a level zero record and is identified as such in the codebook. The first record subordinate to the family record is the member record which is identified as a level 1 type record in the codebook.

Since the purchase record occurs one or more times for each occurrence of the member record, it has a level number of 2. Each member record must be followed by at least one purchase record. Two successive member records without at least one purchase record in between means that the hierarchy is incomplete.

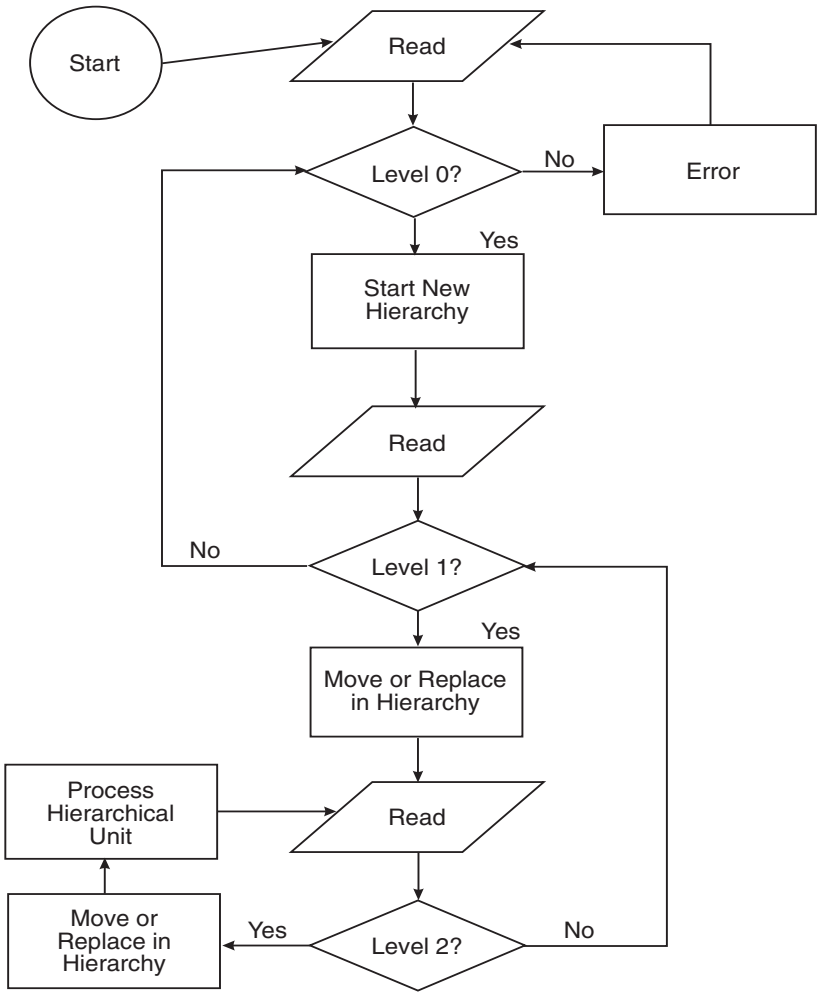
Each level of the hierarchy can consist of more than one record type, but there must be only one of each type in succession at that level. In the following illustration member information spans over two records and family information spans over three records.



The first record type of each hierarchical level must be uniquely identified to TPL REPORT by specifying a unique record value, of any length, in the codebook. In this way missing levels can be detected when the data file is read. If a level is skipped (e.g., LEVEL 0 to LEVEL 2), an error message will indicate that there are records that are not in the expected sequence. When the next record of the highest level (e.g. LEVEL 0) is found, processing will resume.

Below is a flowchart showing the tests made for a three-level hierarchical file.

General Processing for a Three-Level Hierarchical File



Codebook Entries

The first record of each level of a hierarchical file must contain a record identifier by which the record can be uniquely identified. This record identifier follows the key word MARKER in the codebook RECORD clause and is a data name which must be described somewhere within the record description. The data name must be the name of a control variable, and the value following the equal sign must be within quote marks. The record MARKER must be accompanied by a record LEVEL number. (For additional details, see the Record Name Clause section of the chapter called **Codebook**.

```
BEGIN HIERARCHY CODEBOOK

FAMILIES RECORD MARKER FMID ='A' LEVEL 0
    FMID CON 1    ( = 'A' )
    REG CON 1
      (
         'Northeast'  =  1
         'Midwest'    =  2
       )
    JOB CON 1 (1:9)
    AGE CON 2
    CONDITION LABEL IS VALUE
      (    16:99    )
    PERSONS OBS 2
    INCOME OBS 5

MEMBER RECORD MARKER MBID='B' LEVEL 1
    MBID CON 1 (='B')
    AGE_M OBS 2
    SEX CON 1
      (
         'Male'       =  1
         'Female'     =  2
      )
    OCCUPATION OBS 3
    FILLER 5

PURCHASES RECORD MARKER PRID ='C' LEVEL 2
    PRID CON 1 (='C')
    ITEM CON 1
      (
         'Bread'      =  1
         'Fish'       =  2
         'Milk'       =  3
         'Eggs'       =  4
      )
    COST OBS 3
    DAY  CON 1
      (
         'Day 1'      =  1
         'Day 2'      =  2
         'Day 3'      =  3
      )
    PKG_T CON 1 (1:2)
    FILLER 5

END HIERARCHY CODEBOOK
```

How Hierarchies Interact with TPL REPORT Statements

The following sections explain how each TPL statement reacts with a hierarchical file. First, you should think of all records which make up a hierarchical unit (level 0 through a single occurrence of the lowest level) as being read into one contiguous area. The result may be thought of as one long record representing a processing unit. After this unit is processed, another record will be read. If this record belongs to the lowest level of the hierarchy it will replace the preceding record having the same level number, and the resulting new hierarchical unit will be processed again.

If a record is read which belongs to a higher level of the hierarchy (lower level number), that level will be replaced, and successive reads will replace all lower level records until another hierarchical unit is formed. Incomplete hierarchical units will be recognized and an appropriate diagnostic message issued. Normally, only complete hierarchical units will be processed. (See the section on Using Incomplete Hierarchies if you need to use data from incomplete units.)

Record Names and the Built-in Variable COUNT

With a hierarchical file, each **record name** variable from the codebook will count records at its hierarchical level.

The built-in observation variable **COUNT** keeps track of the over-all record count for the data file, without regard to hierarchical levels. In reports, it is treated like a variable at the lowest level of the hierarchy.

REPORT Statement

For our examples, we will use consumer responses regarding hotel service. For each consumer, there is a record called PERSON with information about the person and the hotel room. Following each PERSON record are a series of RESPONSE records containing the person's responses to questions about hotel service. Each question and answer is in a separate RESPONSE record. Following are the data records and codebook for this file.

Data

```
116m 85        Person
2Bed     p4       Response
2Food    b1       Response
2Service p4       Response
128f120        Person
2Bed     g7       Response
2Food    b3       Response
2Service g7       Response
145m 90        Person
2Bed     b2       Response
2Food    f5    Response
2Service f5       Response
```

Codebook

```
begin hotel codebook

person 'Person' record level 0 marker m = '1'
m con 1 (= '1')

age 'Age' con 2 (16:45)
sex 'Sex' con 1  (='m' ='f')
room_fee 'Room Fee' MASK '$'999 obs 3

response 'Response' record level 1 marker n = '2'
n con 1 (= '2')

question 'Question' CHAR 7

answer 'Answer' con 1
('Good' = 'g'
 'Poor' = 'p'
 'Fair' = 'f'
 'Bad'  = 'b'
)
rating 'Rating' obs 1

end hotel codebook
```

Report outputs depend on the levels used in the REPORT statements.

Reports Using a Single Level of the Hierarchy

If all variables in the REPORT statement are from the same level, there will be one row for each record at that level.

In the following example from our two-level hierarchy, all of the variables in the REPORT statement are from the PERSON level. There are 3 PERSON records, so there are 3 rows in the report, one for each person.

Example

```
REPORT H1 'Responses to Hotel Questionnaire on Service':
M THEN PERSON THEN AGE THEN SEX THEN ROOM_FEE;
```

Responses to Hotel Questionnaire on Service

Row	M	Person	Age	Sex	Room Fee
1	1	1	16	m	$85
2	1	2	28	f	$120
3	1	3	45	m	$90

Note The built-in observation variable **COUNT** is treated like a variable at the lowest level of the hierarchy. Thus, if you include it in a report that otherwise mentions only variables at a higher level, you will get a report that follows the rules described below for reports using multiple levels of the hierarchy. There will be one report row for each lowest level record so that COUNT can be displayed.

Reports Using Multiple Levels of the Hierarchy

If variables from a lower level are used, there will be one row for each record at the lower level. In this case, any higher level variables will be joined to the information of each lower level record.

In the following example from our two-level hierarchy, the REPORT statement references variables from both the PERSON level and the RESPONSE level. RESPONSE is the lowest level, so the report has 9 rows, one for each response. *Note* that the person information is repeated for each response that comes from that person.

Example

```
REPORT H2 'Responses to Hotel Questionnaire on Service':
M THEN PERSON THEN AGE THEN SEX THEN ROOM_FEE THEN
N THEN RESPONSE THEN QUESTION THEN ANSWER THEN RATING;
```

Responses to Hotel Questionnaire on Service

Row	M	Person	Age	Sex	Room Fee	N	Response	Question	Answer	Rating
1	1	1	16	m	$85	2	1	Bed	p	4
2	1	1	16	m	$85	2	2	Food	b	1
3	1	1	16	m	$85	2	3	Service	p	4
4	1	2	28	f	$120	2	4	Bed	g	7
5	1	2	28	f	$120	2	5	Food	b	3
6	1	2	28	f	$120	2	6	Service	g	7
7	1	3	45	m	$90	2	7	Bed	b	2
8	1	3	45	m	$90	2	8	Food	f	5
9	1	3	45	m	$90	2	9	Service	f	5

Vertical lines have been added to this report to show which variables are from which level of the hierarchy. You do not need to group them by level in your REPORT statements. If you use ALL or OTHER in the REPORT statement, the variables that go into the report from the ALL or OTHER categories will be displayed in alphabetical order by name without regard to hierarchical level.

Comparison of Record Name Values and COUNT

If we add the built-in observation variable **COUNT** to our table, we can see that its value is always reported as the current record number of the most recent record read. The **record name** columns, labeled 'Person' and 'Response' contain the current record count for their respective record types.

Example

```
REPORT H2_C 'Responses to Hotel Questionnaire on Service'
                ' with a Column Added for the Variable COUNT':
COUNT THEN
M THEN PERSON THEN AGE THEN SEX THEN ROOM_FEE THEN
N THEN RESPONSE THEN QUESTION THEN ANSWER THEN RATING;
```

Responses to Hotel Questionnaire on Service with a Column Added for the Variable COUNT

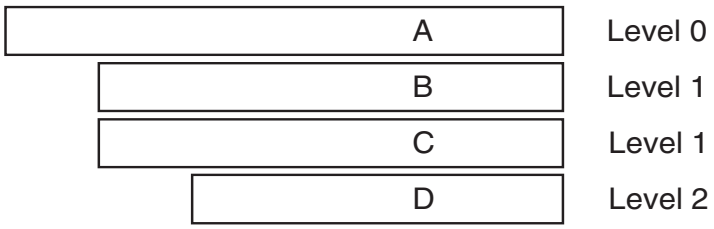
Row	**Count**	M	**Person**	Age	Sex	Room Fee	N	**Response**	Question	Answer	Rating
1	**2**	1	**1**	16	m	$85	2	**1**	Bed	p	4
2	**3**	1	**1**	16	m	$85	2	**2**	Food	b	1
3	**4**	1	**1**	16	m	$85	2	**3**	Service	p	4
4	**6**	1	**2**	28	f	$120	2	**4**	Bed	g	7
5	**7**	1	**2**	28	f	$120	2	**5**	Food	b	3
6	**8**	1	**2**	28	f	$120	2	**6**	Service	g	7
7	**10**	1	**3**	45	m	$90	2	**7**	Bed	b	2
8	**11**	1	**3**	45	m	$90	2	**8**	Food	f	5
9	**12**	1	**3**	45	m	$90	2	**9**	Service	f	5

When Row 1 is complete, 2 records have been read, Person 1 and Response 1, so the current value of Count is 2. When Row 2 is complete, 1 additional Response record has been read, so Count is 3. By the time we get to Row 9, a total of 3 person records and 9 response records have been read, giving a Count a value of 12.

SELECT Statement

The result of a SELECT statement is that either the entire hierarchical unit is selected for processing by following TPL statements, or the next hierarchical unit is formed and tested again. Individual records of a hierarchical unit are never selected for processing alone, even though all variables tested may belong to an individual record.

Assume that a hierarchical file has the following structure.



If a variable is tested in record **D** and does not meet the SELECT condition, a new hierarchical unit is formed by replacing **D** with the next **D**, if one exists. This new hierarchical unit is tested again.

If a variable is tested in record **B** or **C** (both at level 1) and does not meet the SELECT condition, a new hierarchical unit is formed by reading past all following **D** records until a new pair of **B** and **C** records plus a new **D** record is found. This new hierarchical unit is tested again.

If a variable is tested in record **A** and does not meet the SELECT condition, a new hierarchical unit is formed from the next **A**, **B**, **C**, and **D** records. This new hierarchical unit is tested again.

COMPUTE Statement

A computed variable is assumed to belong to the lowest level record of the hierarchy which contains a referenced variable in the COMPUTE statement. For example, if a COMPUTE statement references only variables in level 0 of a hierarchical file, then the computed variable will be assumed to belong to level 0. If a COMPUTE statement references variables in levels 0 and 1 of a three level hierarchy, then the computed variable will be assumed to belong to level 1. If variables in levels 0 and 2 are referenced, the computed value will become part of level 2. If the computation consists only of a literal value, the computed variable will be associated with level 0.

Assume a hierarchical file consists of a family characteristics record at level 0 followed by one or more member characteristics records at level 1. If the family record contains number of rooms in the household (ROOMS) and family size (PERSONS), the statement:

Example

```
COMPUTE  ROOMS_PER_PERSON = ROOMS / PERSONS;
```

will associate ROOMS_PER_PERSON with the family record. A report can then be produced showing the average rooms per person for each family.

Conditional Compute Statement

Like the COMPUTE Statement, the conditionally computed variable is assumed to belong to the lowest level record which contains a referenced variable (control or observation) in the Conditional Compute statement. For example:

Example

```
COMPUTE NEW_WEIGHT =
        1                IF WEIGHT=4;
        PERSON_WEIGHT    IF OTHER;
```

If WEIGHT were at level 0 and PERSON_WEIGHT were at level 1, NEW_WEIGHT would always be assigned to level 1.

RECODE Statement

The RECODE variable may be assumed to apply to the record containing the old variable value, whether the old variable is in the codebook or computed.

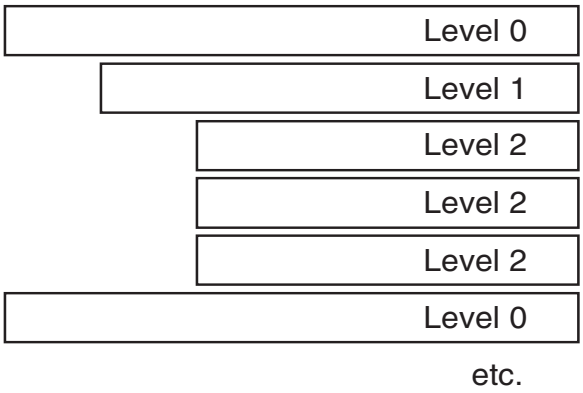
Using Incomplete Hierarchies

Default Treatment

Normally, a hierarchical unit processed by TPL REPORT must be complete. For example, a three level hierarchy cannot consist of only Level 0 records and Level 2 records. Before any processing is done on a hierarchical unit, at least one record must be present at each level. Incomplete hierarchies are reported as errors. If any level is missing, all subsequent records are discarded until a new record is found at the highest level (i.e. the lowest level number).

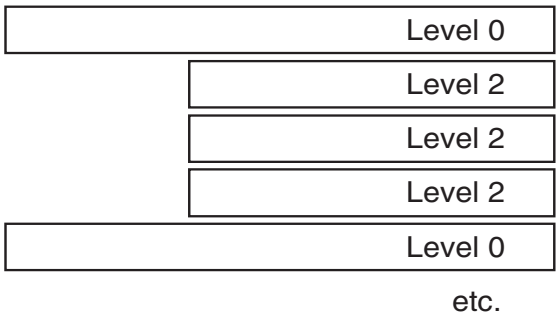
The following sequence of records shows a complete hierarchy.

Complete Hierarchy

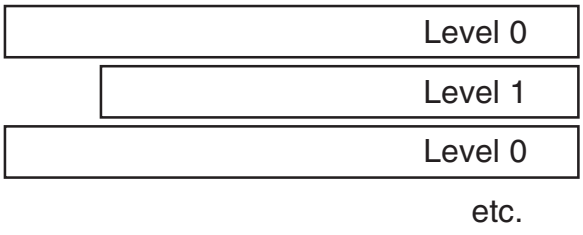


Examples of Incomplete Hierarchies

1. Missing Level 1



2. Missing Level 2



Incomplete hierarchy messages show where the problem occurred in the data file and where processing was resumed. For example:

*** ERROR: Records out of sequence. Level 2 expected for record 6

*** ERROR: Recovered from level error on record 8

Forcing Incomplete Hierarchies to Be Included in Reports

The statements

```
TABULATE INCOMPLETE HIERARCHIES = YES;  (NO is the default)
```

and

```
REPORT INCOMPLETE HIERARCHIES = NO;  (YES is the default)
```

can be used to control the treatment of incomplete hierarchies. By choosing TABULATE INCOMPLETE HIERARCHIES = YES; you can use data from higher level records even though records are missing at the lowest levels.

Note that records cannot be missing from middle levels. For example, a Level 1 record cannot be included if it is followed by a Level 3 record.

The INCOMPLETE HIERARCHIES statements can be entered either in the codebook after the BEGIN CODEBOOK statement or in the report request after the USE statement. A statement in the report request will override any conflicting statement entered in the codebook.

Example of Statements in Codebook

```
BEGIN hierarchy CODEBOOK

TABULATE INCOMPLETE HIERARCHIES = YES;

families RECORD MARKER achar = 'A' LEVEL 0

  FILLER 4

  achar CON 1                 /* The marker field.  'A' is the only      */
        ( = 'A' )             /* valid value for this type of record.   */

  month OBS 2                 /* The month of the survey.               */
        .
        .
```

Example of Statements in a Report Request

```
USE hierarchy CODEBOOK;

TABULATE INCOMPLETE HIERARCHIES = YES;
REPORT INCOMPLETE HIERARCHIES = NO;

RECODE quarter ON month;
        '1st Quarter' if 1:3;
        '2nd Quarter' if 4:6;

REPORT toplevel:  quarter then ..... ;
```

Following is the data used in earlier examples in this chapter. All hierarchies are complete, because each PERSON record has at least one subordinate RESPONSE record.

```
116m 85         Person
2Bed     p4       Response
2Food    b1       Response
2Service p4       Response
128f120         Person
2Bed     g7       Response
2Food    b3       Response
2Service g7       Response
145m 90         Person
2Bed     b2       Response
2Food    f5       Response
2Service f5       Response
```

We can make the first hierarchy incomplete by removing its RESPONSE records as follows:

```
116m 85         Person with no Responses
128f120         Person
2Bed     g7       Response
2Food    b3       Response
2Service g7       Response
145m 90         Person
2Bed     b2       Response
2Food    f5       Response
2Service f5       Response
```

We can include the first PERSON record in reports by adding TABULATE INCOMPLETE HIERARCHIES; to the codebook or report request. In this case, we will add it to the report request.

Example

```
USE HOTEL CODEBOOK;

TABULATE INCOMPLETE HIERARCHIES = YES;

REPORT H3 'Responses to Hotel Questionnaire on Service':
M THEN PERSON THEN AGE THEN SEX THEN ROOM_FEE THEN
N THEN RESPONSE THEN QUESTION THEN ANSWER THEN RATING;
```

Responses to Hotel Questionnaire on Service

Row	M	Person	Age	Sex	Room Fee	N	Response	Question	Answer	Rating
1	1	1	16	m	$85	(d)	0		(d)	(d)
2	1	2	28	f	$120	2	1	Bed	g	7
3	1	2	28	f	$120	2	2	Food	b	3
4	1	2	28	f	$120	2	3	Service	g	7
5	1	3	45	m	$90	2	4	Bed	b	2
6	1	3	45	m	$90	2	5	Food	f	5
7	1	3	45	m	$90	2	6	Service	f	5

The values for the first PERSON record are in the first row of the report. Since it has no RESPONSE records, no RESPONSE values can be shown. The values are displayed as (d) or blank, depending on the variable type. The **control and observation** variables, N, Answer and Rating are shown as **(d)**; the **CHAR** variable Question is shown as **blank.** Note that the **record** variable Response is shown as **0,** because no Response record has been found for the first person.

Interaction with SELECT Statement

If your report request has a SELECT statement that references one or more *variables on the level of a missing record*, that part of the SELECT statement *will have no effect*. In the preceding report, if we had:

```
SELECT IF SEX = 'm' AND ANSWER = 'p';
```

the Person record with no lower level records would be selected on the basis of SEX = 'm' even though it has no lower level records to satisfy ANSWER = 'p'.

Message Suppression

By default, error messages will be reported when incomplete hierarchies are encountered. To suppress incomplete hierarchy messages, use the statement

```
REPORT INCOMPLETE HIERARCHIES = NO;
```

CHAPTER 14

Repeating Groups

VARIABLES THAT REPEAT WITHIN RECORDS

Introduction

When one variable or a collection of variables repeat within a record, they can be described as a repeating group.

One example of a repeating group is a time series in which each data record contains a sequence of 12 values, one for each month of the year. Another example would be a survey questionnaire that contains a series of questions where each question has the same set of possible responses.

The repeating group feature lets you describe the repeating unit only once in the codebook and assign a name to it. You can also assign a name and/or label to each repetition so that the repeating group variable looks like a control variable with the same number of values as the number of repetitions in the group.

Restrictions on the Use of Repeating Groups in Report

Repeating group data can be displayed in reports, but there are currently two cases in which you should not try to do reports with repeating groups.

1. Multiple repeating groups in the same report, for example, if there are multiple groups described in a codebook and you enter the statement:

   ```
   REPORT G1:  ALL;
   ```

2. Combinations of repeating groups and hierarchies in the same data file. This combination could result in abnormal termination of your report job.

Describing Repeating Groups in the Codebook

Format

```
BEGIN GROUP  group-name ['print label']
        [REDEFINES var-name]  REPEATS n

[(Name1 and/or Label1, Name2 and/or Label2, ...
        Name-n and/or Label-n)]

    elementary-item description(s) follow

END GROUP  group-name
```

The commas are required between group repetition names, labels, or name/label pairs. The repetition value, n, must have a value of 1 or more.

To describe a repeating group continuation, use:

Format

```
CONTINUE GROUP  group-name

    elementary-item description(s) follow

END GROUP  group-name
```

The following rules and comments apply to the description of repeating group variables in the codebook.

1. The REPEATS clause must have a value of 1 or more.

2. Within a repeating group there must be at least one elementary item. The elementary items can be control, observation, char or filler. In addition, groups can be contained within groups. We refer to this situation as "nested" repeating groups.

3. **The repeating group name is a control variable** which takes the values of 1 through n, where n is the repetition value. Each repetition can have an optional name and/or print label. If a name is provided for a repetition, but no label is provided, the name will be used as a print label. If no name or label is provided for a repetition, the label "n group-name" will be assigned to that occurrence.

4. Repeating groups can appear anywhere in codebooks, except that they cannot span across data records of different types.

5. Group variables can be redefined and group variables can redefine other variables.

6. The CONTINUE GROUP clause describes the situation where all fields of the group are not stored side by side. Instead, the repetitions of one or more fields follow after all of the repetitions of the field(s) in the location where the group is first defined.

How Repeating Groups Interact with TPL REPORT Statements

Record Names, Group Names and the Built-in Variable COUNT

R**ecord name** variables from the codebook have the usual meaning. Each record name variable contains the count of records of its type. For each record, **repeating group** variables count repetitions if their groups within the record.

The built-in observation variable **COUNT** keeps track of the over-all record count for the data file, without regard to hierarchical levels or repeating groups.

REPORT Statement

For our examples, we will use the following four data records and the codebook describing these records. Each record contains information about a household: a city code followed by a group of evaluations of city services. There are three repetitions of the group, one for Police Protection, one for Library Services and one for Street Maintenance. The evaluations are coded for **G**ood, **F**air, **P**oor or **blank**.

Data

```
01 GFP
01 GFP
02 PFG
03 FF (the last character in this record is a blank for "No Response")
```

Codebook

```
BEGIN GROUP_CB

HOUSEHOLD 'Household' MASK 999 RECORD
```

```
    CITY 'City'  CON 2
    (
    'Concordia' = 1
    'Frostburg' = 2
    'Silver Spring' = 3
    )
  FILLER 1

BEGIN GROUP  SERVICE 'Service'  REPEATS  3
   ('Police Protection',
   'Library Services',
   'Street Maintenance')

      EVALUATION 'Evaluation'  CON 1
      ('Good' = 'G'
       'Fair' = 'F'
       'Poor' = 'P'
       'No Response' = ' '
      )
END GROUP  SERVICE

END GROUP_CB
```

We can view repeating groups as sub-records or records withing a record. Report outputs depend on whether we use only variables from the record level or we use variables from repeating groups, with or without variables from the record level.

Reports that Do Not Use the Group Variables

If all variables in the REPORT statement are from the record level, there will be one row for each record. The report will not be affected by any repeating groups that may be described in the codebook.

Reports that Use One or More Variables from a Repeating Group

If variables from a repeating group are used, there will be one row for each repetition of the group. In this case, any record level variables will be joined to the information of each group repetition.

In the following example using our small data file and codebook, the REPORT statement references variables from both the record level and the repeating group called SERVICE. The report has 12 rows, since each of the 4 records has 3 group repetitions. *Note* that the HOUSEHOLD information is repeated for each repetition of SERVICE evaluation.

Example

```
REPORT G1 'Report using the repeating group called SERVICE.':
    HOUSEHOLD  CITY  SERVICE  EVALUATION;
```

Report using the repeating group called SERVICE.

Row	Household	City	Service	Evaluation
1	1	01	1	G
2	1	01	2	F
3	1	01	3	P
4	2	01	1	G
5	2	01	2	F
6	2	01	3	P
7	3	02	1	P
8	3	02	2	F
9	3	02	3	G
10	4	03	1	F
11	4	03	2	F
12	4	03	3	

Horizontal lines have been added to this report to separate the report rows from each of the 4 records. Note that the column for the group variable called SERVICE contains the repetition numbers for the group.

If you use ALL or OTHER in the REPORT statement, the variables that go into the report from the ALL or OTHER categories will be displayed in alphabetical order by name without regard to whether they are at the record level or contained in a group.

Using Repetition Values and Labels

Group repetitions can be referenced by repetition number the same as if these numbers were values of a control variable.

Labels for group repetitions can be displayed in report cells in the same way as condition labels for control variables. In the following example, we use a FORMAT statement with the Report G1 shown above. The FORMAT statement replaces values with labels for all of the control variables in the report, including the group variable called SERVICES.

Example

```
FOR VARIABLES CITY, SERVICE AND EVALUATION:
        USE CONDITION LABELS;
```

Report using the repeating group called SERVICE.

Row	Household	City	Service	Evaluation
1	1	Concordia	Police Protection	Good
2	1	Concordia	Library Services	Fair
3	1	Concordia	Street Maintenance	Poor
4	2	Concordia	Police Protection	Good
5	2	Concordia	Library Services	Fair
6	2	Concordia	Street Maintenance	Poor
7	3	Frostburg	Police Protection	Poor
8	3	Frostburg	Library Services	Fair
9	3	Frostburg	Street Maintenance	Good
10	4	Silver Spring	Police Protection	Fair
11	4	Silver Spring	Library Services	Fair
12	4	Silver Spring	Street Maintenance	No Response

CHAPTER 15

Labels

CREATING AND FORMATTING PRINT LABELS

Any variable can have a print label associated with it. The print label follows the variable name when the variable is described in the codebook or created in a report request. When the variable is used in a report, this label will print in place of the original variable name. Other report elements that can have print labels are listed below. Two important report elements, **report titles** and **labels assigned to values** in RECODE statements, are print labels and can contain any of the formatting elements described in this chapter.

If you do not specify print labels, default labels will be created for reports. Default labels are satisfactory for identifying the contents of a report, but you may wish to specify your own labels to make them more informative or to take advantage of some of the label formatting options.

This chapter describes all of the formatting options you can use in individual labels. For a description of default treatments, see the "Report" chapter.

A typical label consists of text that is bounded by single or double quote marks. The text can include spaces, upper and lower case letters, and special characters. Many formatting options are available for print labels. Break points can be chosen for multiline labels, and alignment can be specified. If you are working in PostScript mode, you can vary the type styles within labels by inserting font specifications.

Print label options apply to all of the following report elements:

1. Records described in the Codebook
2. Control variables
3. Control variable values
4. Observation variables

5. Char variables
6. Report titles
7. New values created with RECODE statements
8. Subtotals and grand totals

Automatic Print Labels

When print labels are not specified, they are automatically created according to the following rules:

Observation and Char Variables

If no label is assigned to an observation variable, the variable name is used as the print label. This rule applies to variables that are described as RECORD, OBS or CHAR in the codebook and to variables that are computed in a report request.

Control and RECODE Variables and Their Values

If no label is assigned to a control variable from the codebook or a RECODE variable created in a report request, the variable name is used as the print label. See the statement USE VARIABLE NAME to specify that a name should be used regardless of the presence another label.

If no labels are assigned to the values of a control variable, labels are generated from the condition names, if present, or from the values themselves. You can display labels in place of values in report cells. See the RECODE statement and the USE CONDITION LABEL statement for ways of doing this.

Report Titles

If no report title is assigned in the REPORT statement, the report name is used as the title.

Subtotals and Grand Totals

If no label is assigned, default labels are used. See the "Totals" chapter for defaults.

Whenever a name is used as a label, any letters used in the name are printed in upper case. For example, if the observation variable called Income is not followed by a print label, the name INCOME will be used as the default print label. If the name contains underscore (_) characters, they will be replaced with blanks when the name is printed. For example, the name Average_Income will print as **AVERAGE INCOME**.

Creating Your Own Print Labels

Labels can be created in the codebook or report request. They can also be created or replaced using REPLACE statements in a FORMAT request.

A simple label consists of a text string surrounded by single or double quotes. An example of a simple label assigned in a codebook is:

```
INCOME 'Annual Income in Thousands' OBS 5
```

When the variable INCOME is used in a REPORT statement the label **Annual Income in Thousands** will be used to identify the INCOME column in the report.

In the codebook, print labels can optionally be included following variable names and can be assigned to control variable values. Examples of print labels assigned in a codebook are:

```
FAMILIES 'Family Count'  RECORD
AMT_WK 'Dollars spent per week' OBS 7
AUTO 'Automobile owned?'  CON 1
  (
        YES 'Yes'  = 1
        NO 'No'  = 2
  )
HEADS_WORK  'Class of work of family head' CON 1
  (
        ' White Collar'  = 1
        ' Blue Collar'  = 2
        ' Other'  = 3
  )
```

Within a report request, any of the TPL statements that create new variables can optionally include print labels. A print label can also follow the report name in a REPORT statement, in which case that print label will be used as the report title. The following examples show uses of print labels within a report request.

```
COMPUTE INCOME 'Total Family Income' =
         HEAD_INCOME + OTHER_INCOME;

RECODE INC_CL 'Income Classifications' ON INCOME;
        'Less than $5,000'       IF < 5000;
        '$5,000 to $10,000'      IF 5000 :10000;
        'Above $10,000'IF OTHER;

SUBTOTAL MONTH_TOT 'Month subtotals' ON MONTH  REPORT T3;
DISPLAY ALL;
```

```
REPORT FAM_DAT 'Family Income Classifications' :
    INC_CL THEN INCOME THEN STATE;
```

Characters Allowed in Label Strings

With only a few exceptions, label strings can contain any character that is available on your keyboard. The quote and backslash (\) characters must be entered in a special way as described in the next section.

We recommend that you not enter tabs or carriage returns (typed with the **<Enter>** key) in label strings. Tabs will be printed as blanks, and carriage returns will be removed before printing. You can get the effect of a tab at the beginning of a label by using the INDENT option described later in this chapter. If you are entering a label string that is longer than one line, you can break it into multiple segments by ending each line with a quote, typing **<Enter>**, and continuing on the next line beginning with another quote. The segments will be joined when the label is printed.

If there are characters available on your printer that are not on your keyboard, you can enter them in label strings either by using a character name or code. A character name is preceded by **&** and followed by **;** For examples **É** represents an E with an acute accent above it. Character names are case sensitive. **é** represents **e** with an accute accent above it.

Character codes are entered by typing **\nnn** where **nnn** is the 3 digit decimal code for the character. Three digits are always required. If the character can be represented by fewer than 3 digits, add leading zeros. For example, for a character represented by the code 65, enter **\065**.

The value nnn must be the DECIMAL code for the character. The character code tables in some software manuals show the octal or hexidecimal codes for the characters. If you are referring to such a table, you must convert the code to its decimal equivalent. Character set tables showing decimal codes and character names are included at the end of this manual in an appendix.

Quotes and Backslashes in Labels

Since quotes are used to show the beginning and end of a label string, they must be entered in a special way if they are to be used inside a label string. If single quotes are used at the beginning and end of the label string, two successive single quotes are required to print one single quote within the label. If double quotes are used at the beginning and end of the label string, two successive double quotes are required to print a double quote within the label.

String expression	Will print as
'"FIRST EXAMPLE"'	'FIRST EXAMPLE'
'USER"S CHOICE'	USER'S CHOICE
"User's Choice"	User's Choice
'BUT "LESS THAN" 100'	BUT 'LESS THAN' 100
"40 BUT LESS THAN 60"""	40 BUT LESS THAN 60"
""""	' (a single quote inside double quotes)
'"'	" (a double quote inside single quotes)
' '	(a blank inside single quotes)

The backslash (\) character has a special use for entering characters that are not on the keyboard. If you want to include a backslash character in a label, enter a double backslash. For example, the label string **'\\In Thousands\\'** will print as:

\In Thousands\

Label Length

Label length is virtually unlimited. The practical limit on the length of a label is imposed by the requirement that there must be room for at least one line of data on each page of a report. In other words, if a label is so long that it takes up a whole page, there will be no space left for anything else.

The Null Label

The null label consists of two consecutive quote marks with nothing between them. When a null label is assigned to a report or a variable used in a report, the label space will be blank.

Null labels are most useful for suppressing values in RECODE statements. For example, if we want to include family income values in a report, but suppress the display of values greater than $99,999, we can recode income to contain original values or null values:

```
RECODE REP_INC ON INCOME;
"         IF  > 999999;
VALUE IF  OTHER;
```

Labels with Multiple Segments

The print label can be expressed as a single label string or as two or more segments separated by at least one space, as in:

```
'ALL NONMANUFACTURING'  ' INDUSTRIES FOR 1985-90'
```

When multiple segments are used, they will be interpreted as one label combining the individual components. No space will be inserted to separate the merged segments, so for each pair of segments, a space must be included to separate words. One advantage of this format is that each segment can be entered on a separate line, although the segments will be merged as one continuous label. Another advantage is that label formatting options can be inserted between label segments.

Control of Label Breaks

If a print label is too long for its allotted space, it will be automatically divided over two or more lines. If you want more precise control over label break points, you can use two special formatting options.

Slashes

The first formatting option is provided by the use of the slash (/) symbol. A slash inserted between two label segments will cause the second segment to start on a new line. Each additional slash will cause the insertion of one blank line. Each slash at the beginning or end of a label will cause one blank line to be inserted.

Single slashes cause single spacing between segments. Multiple slashes cause additional line spacing between segments. For example, three slashes separating two print labels would cause triple spacing between them. The expression,

```
'Row One'/'Continue'/'Continue'//'Row Two'
```

would print in a report title as:

```
Row One
Continue
Continue
(space)
Row Two
```

Each segment of the title would be left justified within the report width unless an alignment keyword, such as **CENTER**, is included in the title. In that case each segment would be centered within the report width.

In the following RECODE example. the variable label, **Industry Types**, will have one line of blank space below it. The three assigned value labels will be displayed in the report cells in place of the original values and each will begin one line below its normal starting line.

```
RECODE IND_LABLS  'Industry Types' / ON  INDUSTRY
        /'Manufacturing'  IF  'A';
        /'Non-Manufacturing'  IF  'B';
        /'Farming'  IF  'C';
```

Conditional Hyphens

The second formatting option allows you to specify where the label should break if it is too long for the available space. This conditional hyphenation is best illustrated by an example.

```
'MANU'-'FAC'-'TUR'-'ING'
```

If there is enough space to print all of the components as one consecutive string, they will appear as:

```
MANUFACTURING
```

If there is enough room for only the first seven characters plus a hyphen they will appear as:

```
MANUFAC-
```

with **TURING** appearing on the next line. If only five spaces are available, **MANU-** will appear on one line. **FACTURING** will next be considered for the following line and segmented in the same way if necessary.

When a hyphen at the end of a label segment is followed by a conditional hyphen and the label breaks at that point, only one hyphen will be displayed in the label.

Example For a column of width 10, the label 'Never-'-'Married' will be printed as:

```
Never-
Married
```

Hierarchy of Label Break Points

Labels are divided into multiple lines according to the following priorities.

1. Unconditional Break ('segment' / 'segment')
2. Blank within label string ('segment segment')
3. Hyphen within label string ('segment-segment')
4. Conditional hyphen ('segment' - 'segment')

If none of the above break points are found, the label will be broken at points that allow the segments to be printed with hyphens at the break points.

Label Alignment

LEFT, RIGHT and CENTER

The words LEFT, RIGHT and CENTER can be used with a label to override the default alignment. Default alignment for different types of labels can be changed with ALIGN statements as described in the FORMAT chapter, but a specification of LEFT, RIGHT or CENTER in an individual label will always take precedence.

*******NOTE: For column labels, individual alignment specifications have not yet been implemented in TPL REPORT. See the FORMAT statements ALIGN COLUMN and ALIGN HEADING LABEL to align these labels left or right. *******

Note that the word CENTER can also be spelled CENTRE.

LEFT, RIGHT and CENTER are called **alignment markers**. They can be inserted at the beginning of a label before the first quote, at the end after the last quote, or between label segments if there is more than one segment. For example:

```
REPORT ONE
        LEFT 'ESTABLISHMENT DATA'
        RIGHT 'ESTABLISHMENT DATA' / / /
        LEFT 'Report B-1. Employment data for nonagricultural '
                'establishments by industry.'
         ALL;
```

In this example, the title will be formatted as:

```
ESTABLISHMENT DATA                                  ESTABLISHMENT DATA

Report B-1. Employment data for nonagricultural establishments by industry.
```

As illustrated above, a label can have one or more alignment markers. They affect the label according to the following rules.

1. If you put only one alignment marker in a label, regardless of its location in the label, all segments of the label will take on the specified alignment. For example, the following label with a single marker of RIGHT will be formatted as two lines with both aligned to the right, even though the word RIGHT is placed in the middle of the label.

'All Establishments' / RIGHT 'Reporting this Year'

will print as:

All Establishments
Reporting this Year

2. If there are multiple alignment markers in a label, any label **section** that does not have an explicit alignment marker is assumed to be left-aligned.

   For alignment purposes, the first **section** of a label begins at the beginning of the label. A **section** ends with any of the following label elements: /, RIGHT, LEFT, CENTER, RIGHT IN SPACE, SPACE, and SPACE TO.

   For example, the label

   CENTER 'Workers Compensation' / 'Mining' / RIGHT 'January'

   has the sections:

   'Workers Compensation'
   'Mining'
   'January'

   The section 'Workers Compensation' is centered, because it is preceded by CENTER. It ends with /. The section 'Mining' is left-aligned, because it has no explicit alignment marker. It is ended by both a / and the word RIGHT. The section 'January' is right-aligned, because it is preceded by RIGHT. If this is the report title, it will be displayed as:

   Workers Compensation
   Mining
   January

3. If there are two alignment markers between slashes, or between the beginning and end of the label if there are no slashes, then the sections will be placed on the same line with the specified alignments if there is enough space on the line to do so. If an aligned section doesn't fit, it will be placed on the next line. Consider the following report title:

   REPORT TITLE_SAMPLE
   LEFT 'Workers Compensation Report' RIGHT 'January'

If the report is wide enough for all of the title characters to fit on one line without overlapping, the complete title will be placed on one line with a left-aligned section and a right-aligned section:

```
Workers Compensation Report                                    January
```

If RIGHT and LEFT were reversed in this title as follows:

```
RIGHT 'Workers Compensation Report' LEFT 'January'
```

then 'Workers Compensation' would be right-aligned on one line and 'January' would be left-aligned on the next, since there would never be space for the 'January' following the right-aligned 'Workers Compensation' section.

For another example, suppose a report is 50 characters wide. This means that the title space is 50 characters. The first section of label is left-aligned and takes up 15 characters. The second section is to be centered and takes up 26 characters. The centered section should start at position 25 (the center) - 13 (half the length of the centered segment) = 12. But the first section extends beyond 12, so there is no room for the centered section. Consequently, the centered section appears on a new line.

4. *If you use multiple alignments within the same label, we recommend that you explicitly divide your label into sections that will fit for each line of the label and precede each section with the alignment of your choice. That way, you will always get the expected result.*

Alignment in Page Markers

The FORMAT statement called PAGE MARKER can only have an alignment specified at the beginning (before any label segments, if present). This alignment applies to the entire page marker.

If you want a page marker with part on the left and part on the right, try aligning the page marker LEFT and inserting SPACE TO in front of parts of the marker to "push" them over to the desired location. Some experimenting may be needed to get things in the position you want. An example is:

```
PAGE MARKER = LEFT  SPACE TO 3 cm 'Page ' NUMBER
                    SPACE TO 12.5 cm  'HOUSEHOLD DATA';
```

Note that SPACE TO only applies to left-aligned labels, so this technique can only be used with a left-aligned page marker. Note also that a left-aligned page markers begins at the left margin of the page rather than the left edge of the report below it.

See also RIGHT IN SPACE, described elsewhere in this chapter. This is another option that can help you get a left and right section for a page marker. For example:

```
PAGE MARKER LEFT 'Left marker'
        RIGHT IN SPACE 7.5 IN  NUMBER;
```

In this example, the page width is 8.5 inches. Aligning the page NUMBER right to a location of 7.5 inches puts it at the right margin of the page if the default left and right margin widths of .5 inches are being used.

RIGHT IN SPACE for Right-Alignment to a Selected Point in a Label

A specification of RIGHT in a label causes the following label section to be aligned at the right edge of the label space. If you wish to right-align to some other point within the label space, you can use RIGHT IN SPACE.

Format

```
RIGHT IN SPACE  location  [ unit ]
```

The label **section** following RIGHT IN SPACE will be right-aligned to the **location.** The optional unit of measure can be expressed as inches, cm, or points. If no unit is specified, the unit is assumed to be characters.

The first **section** of a label begins at the beginning of the label. A label **section** ends with any of the following label elements: /, RIGHT, LEFT, CENTER, RIGHT IN SPACE, SPACE, and SPACE TO.

The location is measured from the beginning of the label space. For example, in a report title, the label space begins at the left edge of the report.

Note that RIGHT IN SPACE applies only to left-aligned label sections. Certain types of labels, such as column labels, are centered by default. With these, you must use LEFT to left-align before specifying RIGHT IN SPACE, as shown in the example below.

The location must be within the available space. For example, if you specify the following for a heading label:

```
LEFT  RIGHT IN SPACE 2 INCHES 'Health Insurance'
```

and the column width is only 1.5 inches, the label section can't be aligned to a point 2 inches to the right.

If RIGHT IN SPACE is applied to a label section that cannot fit in the space preceding the location, RIGHT IN SPACE is ignored. For example, if the label

section is 5 inches long, and you specify RIGHT IN SPACE 3 INCHES, the label section cannot fit in the 3 inch space.

Example In the following example, RIGHT IN SPACE is used to right-align two sections of the report title 3 inches into the title space.

```
report one 'Report 86.'
        right in space 3 in 'Plan participation:'/
        right in space 3 in 'Three types of insurance.' :
        health then life then other_ins;
```

Continuation Labels for Report Titles

Report titles are the only print labels that have automatic continuation labels. For example, suppose that we have a report title set up as:

```
REPORT REG_TAB 'Region Summaries for 1981 - 91':  .....
```

If the report continues beyond a single page, a continuation label will follow the title for pages after the first:

```
Region Summaries for 1981 - 91 - Continued
```

The continuation label ' **- Continued**' can be changed or removed using the FORMAT statement **REPLACE TITLE CONTINUATION WITH 'new continuation'**.

Indentation and Spacing in Labels

Changing Label Alignment with INDENT

INDENT specifications can be used in labels to assist in label alignment. If you are working in line printer mode, you can often achieve the desired alignment by adding or subtracting blank characters at the beginning of a label. In some cases, however, you may find it easier to control label alignment using INDENT. This is especially true if you are using the proportional fonts available in PostScript mode. For additional aids to label spacing, see the section called **"Spacing within Labels"**.

Format The format for the indent specification is:

```
INDENT [ + or - ]  amount  [ unit ]
```

where **amount** is the size of the indent (decimal numbers are allowed). The amount can be up to about 25 inches.

The optional **unit** specification can be expressed as inches, cm or points.

Example

```
INDENT .5 INCHES
```

A positive indent amount will shift the label right; a negative amount will shift the label left.

If no unit is specified (as shown in the following example), the unit is assumed to be characters.

INDENT works properly only with left-justified labels.

INDENT applies to all lines of a label that follow it. If you begin a label with INDENT, then add another INDENT specification in the middle of the label, the second INDENT will take effect at the beginning of the next line. For example:

```
indent 1 cm 'label line 1' indent .5 cm / 'label line 2'
```

will give the result:

```
    label line 1
label line 2
```

If slashes are included in the label to show where the label should break to go to a new line, an INDENT specification for the new line can be inserted either before or after the slash. The label

```
indent 1 cm 'label line 1' / indent .5 cm 'label line 2'
```

will give the same result as the label shown above. It is identical except that the INDENT for the second line follows the slash rather than preceding it.

If you have not inserted slashes to show the break point for a long label but wish to control the indentation following the break, you must insert an INDENT somewhere in the label before the break point. For example,

```
INDENT 3 'This is' INDENT 6 ' a long multi-line label.'
```

The first line of the label will be indented 3 characters. Continuation lines will be indented 6 characters.

Indent Restrictions

There must to be space on the current line for at least two characters of label in addition to the indentation.

Indent with PostScript Proportional Fonts

In line printer mode or in PostScript mode with a non-proportional font, all characters, including blanks, are the same width. In PostScript mode, if you are working with a proportional font, the character width depends on the character. Numbers will all have the same width, but for other characters the width will vary. For example, the letter **o** will be wider than the letter **i**. In particular, a blank will take up about half the space of the average character width or the width of a number. If you have specified INDENT in characters, the width used for each unit of indentation will be the same as the width of a number in the font you are using.

If you are working only in line printer mode or in PostScript mode with non-proportional fonts, you can often easily align labels by simply adding blanks to move parts of the label left or right.

If you are working in PostScript mode with proportional fonts, use of INDENT rather than blanks can help you in two ways:

1. You may want to begin work on your report in line printer mode, so that you can display the report on the screen and/or print it on a line printer until you have most of the format characteristics worked out -- then switch to PostScript and proportional fonts. If you adjust label alignment with INDENT rather than by adding blanks, your transition to PostScript mode will be easier. If you use blanks instead, you will need about twice as many blanks to get the same amount of space when you switch to a proportional font.

2. If you need to align an indented label with a particular character in the line above, INDENT will give you finer control than you can get with blanks. You can precisely adjust the label alignment by inches, centimeters or points.

Spacing within Labels Using SPACE and SPACE TO

You can use the words **SPACE** and **SPACE TO** to add a specific amount of space within a label or to space over to a particular location. These spacing options can be used in both line printer and PostScript modes, but they are especially useful with PostScript's proportional fonts, because they let you add a specific amount of space without regard to character sizes.

Format The formats for the two spacing options are:

```
SPACE amount  [ unit ]
SPACE TO amount  [ unit ]
```

where **amount** is the size of the space or the location to "space to". The amounts can contain decimal points for fractional amounts such as 3.5 . The amount can be up to about 25 inches.

The optional **unit** can be expressed as inches, cm or points. If no unit is specified, the unit is assumed to be characters.

The spacing options should only be used with left-aligned label segments. If used in centered or right-aligned segments, they will either be ignored or give results other than what you expect. When SPACE TO is used, the location is always calculated from the start position of the label without regard to indents or blanks that may be included at the beginning of the label.

If a label segment is too long for the current line after space is added, it will be continued to another line with no space at the beginning of the next line.

Examples

```
'Total' SPACE 10 CM 'All Universities'
'Total' SPACE TO 10 CM 'All Universities'
```

In the first example, there will be a space of 10 centimeters between 'Total' and 'All Universities'. In the second example, space will be added between 'Total' and 'All Universities' so that the distance from the start of the label to 'All Universities' is 10 centimeters.

Using SPACE TO and INDENT Together

SPACE TO and INDENT can be combined as shown in the following example where SPACE TO is used to move a portion of the first line of a report title to 1 inch from the beginning and INDENT is used to indent additional lines to the same location. Note that the font change in this title would not affect the non-PostScript format.

Example

```
REPORT S1 LEFT 'Table 3.3e'
SPACE TO 1 INCH INDENT 1 INCH
'Petroleum Imports: Angola, Australia, Bahama Islands, Brazil, '
'Canada, and China.' /
FONT H 10 '(Thousand Barrels per Day)':  ..... ;
```

Table 3.3e Petroleum Imports: Angola, Australia, Bahama Islands, Brazil, Canada, and China
(Thousand Barrels per Day)

Both INDENT and SPACE options are designed to work with left-aligned label segments. All segments of the report title are left-aligned by default, but it is possible to get different alignments for independent segments. The next title is the same as above but the last line is centered. We can make the centering work correctly by setting INDENT back to 0 for the last line so that no indentation is in effect for that line.

Example

```
REPORT S2 LEFT 'Table 3.3e'
SPACE TO 1 INCH INDENT 1 INCH
'Petroleum Imports: Angola, Australia, Bahama Islands, Brazil, '
'Canada, and China.' /
INDENT 0  CENTER  FONT H 10  '(Thousand Barrels per Day)':  ..... ;
```

Table 3.3e Petroleum Imports: Angola, Australia, Bahama Islands, Brazil, Canada, and China
(Thousand Barrels per Day)

PostScript Font Control in Labels

Fonts can be set for different types of labels, including titles, using PostScript FONT statements in the FORMAT language. This method of font selection works well if you want all labels of a certain type to have the same font. Sometimes, however, you may need to use a different type style or size for particular labels or for different sections within the same label. You can do this by including fonts in individual labels. These font specifications take effect only in PostScript mode. They are ignored in line printer mode.

For a complete list of available PostScript fonts, including **bold**, *italic* and underline fonts, see the **FONT** statement in the **FORMAT** section of the manual.

A font specification within a label takes the same form as in the FORMAT language FONT statement. To change the font for an entire label, simply insert the

FONT specification at the beginning. For example, the following label will be printed in **T**imes **B**old **I**talic:

```
FONT TBI '***Revised***'.
```

Fonts can change more than once within a label. For instance, a label could begin with a section of **bold-underlined** type, change to *italic* and end with ***bold-italic***. To change fonts within a label, insert the font specifications anywhere between strings.

The expression FONT RESET can be used at any point to restore the default label font for a later section of the label. The following example shows how FONT and RESET can be used in a title:

```
'As published in '
FONT HI 'Three Little Pigs'
FONT RESET ' by Anon.';
```

The font size is optional. If the font specification is in the middle of a label and does not include a size, the size is the same as for the previous part of the label. If the font specification is at the beginning of the label and does not include a size, the size is the same as the default size for that type of label.

A font remains in effect until another new font is specified or the end of the label is encountered. The special font RESET is the same as the default font for that type of label. Thus, for instance, if we have set the default of **TITLE FONT = H 8**, the example shown above would give the same result as:

```
'As published in '
FONT HI 8 'Three Little Pigs'
FONT H 8 ' by Anon.';
```

In either case, the title would print as:

As published in *Three Little Pigs* by Anon.

The advantage of using the RESET font is that if you change the default font for a particular type of label, you will not need to adjust individual labels to match the new default.

For another example, assume that the default title font has been set with:

```
TITLE FONT H 10;
```

If we want all parts of the title to have the default size, but different styles for some sections, we can add FONT specifications to the title without including sizes. For example,

```
CENTER 'Report B-4.  '
        FONT HBU 'Sales Prices'
        FONT HB ' of New Houses Sold in the United States.'/
        FONT RESET '[Rounded to hundreds of dollars]'
```

This title would print as:

Report B-4. **Sales Prices of New Houses Sold in the United States.**
[Rounded to hundreds of dollars]

If we later find that we need to increase or decrease the size of the title font for all reports, we can do so by changing only the TITLE FONT statement. Size adjustments in the individual titles will be automatic. Assuming that we change the default title font to **TITLE FONT H 8**, the title shown above will print as:

Report B-4. **Sales Prices of New Houses Sold in the United States.**
[Rounded to hundreds of dollars]

If fonts of different sizes are used in a label, the vertical spacing for the label is determined by the largest font, even if the larger font is not used in all lines of the label. For total labels or labels in cells, if the DEFAULT FONT is larger than the label fonts, the spacing will be determined by the DEFAULT FONT.

Superscripts and Subscripts

Superscripts and subscripts can be used in labels. These take effect in PostScript mode only. In line printer mode, the superscript and subscript notations are ignored.

Enter the superscript or subscript notation in the label in front of the appropriate label segments. For superscript, use **SUP** or **SUPER**; for subscript, use **SUB**. The superscript or subscript specification will apply from that point in the label, either to the end of the label or to the next occurrence of the notation **NORMAL**. These notations can be mixed with other label features such as font, spacing and line break specifications.

Superscript characters are raised by the same amount as as superscripted footnote symbols would be; subscripts are lowered to the base line of the label.

Example 'Regular label part ' SUP 'Superscript part ' NORMAL 'End'

The label text 'Regular label part ' will be printed at the normal level, the label text 'Superscript part ' will be raised, and the label text 'End' will be at the normal level.

CHAPTER 16

Masks

FORMATTING DATA VALUES WITH MASKS

For **observation** variables, **values that do not have masks** are rounded to the nearest whole integer and displayed with no special symbols except commas. The values are centered in the report columns. If you want a different format for values, you can specify the format using a print mask.

With a mask, you can format data with decimal points, commas, and special characters such as dollar signs and percent symbols. When a mask is used, data is centered in the report columns based on the size of the mask, or right-alignment can be specified. If you are working in PostScript mode, you can choose the type style for values by inserting font specifications in masks.

A mask can be assigned to any observation variable described in the codebook or computed in a report request. Whenever the variable is used in a report, the mask determines the format for the variable's values. The **REPLACE MASK** statement can also be used in a FORMAT request to assign or replace a mask either by variable or by column. To replace a report value with a different value or some text, see the **RECODE** statement.

The mask functions as a pattern for formatting the values. In its simplest form, it consists of a succession of 9's, one for each digit position of the largest expected value. For example, a mask of:

```
MASK  9999
```

would indicate that the largest expected value has four digits. The values would be centered based on the size of a four digit number and would be printed without commas or other special characters.

Adding Decimal Points and Commas

When decimal points and commas are to be displayed with the values, the symbols are indicated in positions relative to the 9's. The following mask will format the values with a comma and two decimal places:

```
MASK  9,999.99
```

If a value is larger than the mask and the mask contains one or more commas, additional commas will be inserted as required.

A mask cannot end with a comma or a decimal point. If commas are used in a mask, the commas must separate groups of three 9's, starting at the decimal point, if present, or the right-most 9 and proceeding to the left. If there are fewer than three 9's following a comma, they will be ignored. For example, a mask of **999,99** will be treated the same as a mask of **999**.

Decimal values are rounded to the number of decimal places shown by the mask. If a decimal value is formatted without a mask, it is rounded to the nearest integer value.

Rounding Rule

Rounding is done according to the "round even" rule: 5 is rounded up or down depending on the digit to the left of the 5. If the digit to the left of the 5 is even, it rounds down. If the digit to the left is odd, it rounds up. (A blank to the left is considered to be a zero and thus even.)

For example, with a mask of 99.9:

```
5.8500 -> 5.8      (8 is even -> round down)
5.7500 -> 5.8      (7 is odd  -> round up)
```

This rounding rule is part of the IEEE and ANSI standards for binary and floating point arithmetic.

Note that detail values may not add to totals because of rounding. This is true regardless of the rounding rule being used. The following illustrates results with the "round even" rule. In this case, the sum of the rounded detail values is greater than the rounded sum.

```
2.5 -> 2      (2 is even -> round down)
4.5 -> 4      (4 is even -> round down)
------------
7.0  ≠  6
```

Creating Decimal Places

A value is assumed to be a whole number with no decimal places unless:

- it contains values described with a SHIFT LEFT clause in the codebook;
- it contains values described as floating point in the codebook; or
- it contains values resulting from computations that add decimal places (for example, division in a Compute statement).

If the value is assumed to be a whole number and is formatted with a mask that contains a decimal point, the whole number will be printed to the left of the decimal point with 0's to the right of the decimal point. For example, if the mask **99,999.99** is used to display a cents aggregation of 47378, the displayed result will be **47,378.00**, since the decimal point is assumed after the 8.

To show values of this type with the correct number of decimal places, the decimal places must be created by division in a COMPUTE statement. For the dollars and cents example, we can create two decimal places by dividing the values by 100 in a COMPUTE statement as in:

```
COMPUTE DOLLARS USING 99,999.99 = CENTS / 100;
```

The value of **473.78** used with the mask **99,999.99** will then be displayed correctly as **473.78**.

Leading Zeros

When a decimal value less than zero is printed, it is always displayed with a zero to the left of the decimal point. An example is **0.48**. This is true regardless of what mask is used for the value, even a mask such as **MASK .99**. If you do not want to display these zeros, you can remove them by using the FORMAT statement DELETE LEADING ZEROS; If this statement is used, our example value will print as **.48** instead of **0.48**.

$, % and Other Character Strings in Masks

A mask can be preceded or followed by a character string bounded by quote marks. In this case, the character string will be displayed with all values to which the mask applies. For example, if an entire column is to be printed with a trailing percent symbol, a mask such as **99.9'%'** could be used. Likewise, if an entire column is to be printed with a leading dollar sign, a mask such as **'$'9,999.99** could be used.

A mask can consist of only a character string bounded by quote marks. In this case, the character string will be displayed alone without the value. You can even make a report value blank by using **MASK ' '**.

See also the chapter called **RECODE** for additional ways of replacing values with text or combining values with text in reports.

Replacing Rounded Digits with Zeros

Data can be rounded and displayed with trailing zeros by inserting zeros in the mask. For example, a mask of **999,000** causes data to be rounded to the nearest thousand and displayed with three zeros in place of the rounded digits. The value **876859** will be displayed as **877,000**.

Zeros in masks are ignored if they are to the right of a decimal point or if there are any 9's to the right of the zeros. A mask of **9909** is treated the same as a mask of **9999**; a mask of **9900.00** is treated the same as a mask of **9999.99**.

Alignment of Values

Data values for which a mask is given will be centered within the column width unless other alignment is specified. The number of characters making up the mask will be used to control the centering. The mask may be thought of as being positioned at the center of the column, with values being aligned with the mask from right to left. For example, a mask of **'$'99,999** used together with a column width of 10 (including the column divider) would give the following results.

Value	Will display as
23567	$23,567
146	$146

In line printer mode, if a mask cannot be perfectly centered because of an uneven number of spaces, it is adjusted to the right one position. For example, if there are 9 spaces available for a 6 character mask, the mask will be positioned with 2 spaces to the left and 1 space to the right. In PostScript mode, the mask will always be centered.

The keywords **RIGHT** and **CENTER** can be used with a mask to force alignment of values to the right side or center of the column. For example:

```
COMPUTE WK_SALARY 'Weekly Salary'
        MASK '$'999,999 RIGHT = 40 * PAY_RATE;
```

Since the default alignment for masks is **CENTER**, you do not need to add this word to a mask to specify centering.

See also the statements **ALIGN CELLS** and **ALIGN COLUMN** in the **FORMAT** chapter for additional ways of aligning data.

Note ALIGN statements can be applied to any type of variable, not only observation variables. For observation variables, masks and ALIGN statements such as ALIGN COLUMN interact. If both a mask and an ALIGN statement apply to the same column, the last specification wins.

Treatment of Large Values

If a value has more digits than shown in its mask, column spaces to the left of the mask space are used, if available. If there is not enough space, the following steps are taken as required to print the value:

1. The value is aligned to the right regardless of the alignment specified by the mask.
2. Leading and trailing mask strings are removed.
3. Digits to the right of a decimal point are deleted one at a time.
4. If there is still not enough space to print the value, the value is replaced with **(f)** to indicate that it will not **f**it.

PostScript Font Control in Masks

In PostScript mode, the font for data values is determined by the DEFAULT FONT that you have chosen for PostScript report output. Sometimes, however, you may need to use different type styles or sizes for particular variables. You can do this

by including fonts in individual masks. These font specifications take effect only in PostScript mode. They are ignored in line printer mode.

For a complete list of available PostScript fonts, see the **FONT** statement in the **FORMAT** section of the manual.

A font specification in a mask takes the same form as in the FORMAT language FONT statement for masks. To change the font for an entire mask, simply insert the FONT specification at the end of the mask. For example:

```
COMPUTE SALARY MASK RIGHT '$'999,999 FONT TBI 8 = 12 * MO_RATE.
```

Note For a data mask, the FONT specification *must be at the end of the mask* and the FONT applies to the entire data mask.

If you want to use a variety of fonts within a mask, you may be able to get the desired result by using a **RECODE** statement. Fonts can be varied within RECODE value assignments. For example:

```
RECODE AVG_AGE 'Age' ON AGE;
FONT TIU 'Average ' FONT RESET 'Age' FONT HB VALUE IF ALL;
```

The font **size** specification is optional. If size is not specified, the size will be determined by the PostScript DEFAULT FONT.

The vertical spacing of a data row is not adjusted for the font specifications of individual mask fonts. The spacing is set according to the largest font in a RECODE value for the data row OR the DEFAULT FONT -- whichever is larger. If fonts of different sizes are used for different columns and some of the mask font sizes are substantially larger than both the RECODE values and the DEFAULT FONT sizes, it is possible that the data values with large fonts could overlap those above or below.

Sample Report Using Masks

The following report show how various data values would be displayed with different masks.

```
use banking codebook;

select 10;  /* show only the first 10 records */

compute bal_dol center 'Loan'/'Balance'  mask right '$'999,999  = bal;
compute rate 'Effective'/'Interest' mask 'Rate ' 99.99'%' font tiu = regz/100;

report m1: loan_no then bal_dol then rate;
```

M1

Row	Loan Number	Loan Balance	Effective Interest
1	00004	$1,000	*Rate 15.00%*
2	00008	$5,743	*Rate 5.25%*
3	00011	$1,546	*Rate 5.25%*
4	00036	$3,144	*Rate 5.25%*
5	00040	$3,065	*Rate 5.25%*
6	00041	$2,594	*Rate 5.25%*
7	00043	$3,009	*Rate 5.25%*
8	00043	$12,900	*Rate 14.00%*
9	00044	$100,000	*Rate 0.00%*
10	00045	$15,000	*Rate 0.00%*

CHAPTER 17

PostScript

PUBLICATION QUALITY REPORTS USING POST-SCRIPT

PostScript®, developed by Adobe Systems Incorporated, is a printer-independent language for describing a page of text and/or graphics. With TPL REPORT PostScript features, you can create publication-quality reports with choices of type style and size, including proportional type. Because PostScript provides a standard way of describing a page, PostScript reports can be printed on any laser printer or typesetting machine that processes PostScript. In Windows, you can use Ted, the TPL Editor, to print PostScript reports on any of the printers you use.

Even if you do not publish reports, you may wish to take advantage of the PostScript features for the following reasons. First, the reports look like they have been professionally typeset and are very impressive. Second, you can get much more information on a page with proportional type styles. Unless you use upper case letters throughout your labels, the labels will take up less space than they would with a fixed character width.

In TPL REPORT PostScript mode, you can select both the type style and the type size for all kinds of labels, including titles, and for the data values in your reports. With proportional type styles, special formatting is required to get the correct column alignments for a report. TPL REPORT does this special formatting for the entire range of PostScript type styles and sizes. Another special PostScript feature lets you rotate reports for sideways printing.

You can print PostScript reports directly from TPL REPORT or combine them with text using other desktop publishing software. This User Manual provides an example of how the reports can be combined with text. All of the reports shown in this manual were prepared using TPL REPORT in PostScript mode. The text was prepared using an editor (word processing program). Then the text and reports were combined using a desktop publishing system.

Windows Note You can display and print PostScript reports directly from Ted, the TPL Editor, regardless of whether or not you have a PostScript printer. To print the reports in other ways, such as from the command line, you need to have a PostScript printer as described below under UNIX Notes.

UNIX Notes You need to have a PostScript printer to print PostScript reports. If you do not have a PostScript printer, you may still be able to print PostScript reports by adding a PostScript board or cartridge to your computer or printer, or by using a program that translates PostScript for the type of printer you have. Due to the variety of options and the rapidly changing technology, it is difficult for us to make a recommendation here.

PostScript reports created on a UNIX system can also be displayed and printed on a Windows PC using View or Ted, the TPL Editor. Please contact QQQ Software, Inc. if you need one of these programs.

PostScript FORMAT Statements

PostScript options are chosen with FORMAT statements that can be entered in a format request or in a TPL profile. The FORMAT statements that are especially relevant to PostScript are:

POSTSCRIPT to select PostScript format for reports;
FONT to select type style and size;
REPLACE MASK FONT to change only the mask font;
ROTATE for sideways printing;
EXTRA LEADING to control line spacing;
RULE WEIGHT,
DOWN RULE WEIGHT,
RETAIN CROSS RULES WEIGHT = n and
RETAIN SIDE RULES WEIGHT = n to adjust the thickness of rules.
CODEPAGE for non-English alphabets.
DEFAULT COLOR to set the default color (see COLOR Default);
LABEL COLOR to set the color default for labels (see COLOR Default);
RULE COLOR to set the color default for rules (see COLOR Default);
SYMBOL COLOR to set the color default for symbols (see COLOR Default);
COLOR = NO and
REPLACE COLOR color WITH FONT font to substitute fonts for colors;
REPLACE MASK COLOR to change cell color.

These statements are documented extensively in the FORMAT chapter. See also the chapter called "Color" for an introduction to color features.

Getting Started with PostScript

Windows

The installation process sets PostScript defaults in the TPL TABLES system profile so that you can get properly formatted PostScript reports automatically. You can change, delete or override any of the installation defaults with FORMAT statements. See the FORMAT statements, **POSTSCRIPT** and **FONT**, for complete details. If you wish to change from PostScript to ascii text output, you can do so with the FORMAT statement **POSTSCRIPT = NO;**.

UNIX

When you install TPL TABLES, you can choose between PostScript and line printer (ascii text) mode. If you choose PostScript, the installation process will set PostScript defaults in the TPL TABLES system profile so that you can get properly formatted PostScript reports automatically. You can change, delete or override any of the installation defaults with FORMAT statements. See the FORMAT statements, **POSTSCRIPT** and **FONT**, for complete details.

Use of PostScript requires a choice of type style and size. We call this combination of style and size a **FONT**. When TPL TABLES is installed to run in PostScript mode, a set of FONT statements is entered in the TPL TABLES profile. If there are no FONT statements, TPL TABLES will assume a font of **C 12** (Courier 12). This will result in reports that look very much like ascii text reports.

There is only one other thing to consider before getting PostScript output. This is the specification of sizes for pages, margins and report characteristics such as columns.

When using PostScript, you should express page size and margin sizes in something other than characters and lines. This is because, with PostScript, you can choose different character sizes. If page and margin sizes are expressed in characters and lines, the size of the page will vary as the character size changes. This result is usually undesirable.

For example, the following statements will "shrink" the report into the lower left portion of the page:

```
POSTSCRIPT = YES;
PAGE WIDTH = 80;
PAGE LENGTH = 66;
DEFAULT FONT = H 8;   /* Helvetica 8 */
```

If you are using 8 1/2" by 11" inch paper, you can get your report properly positioned on the page by changing the page specification so that it does not depend on character size. For example:

```
PAGE WIDTH = 8.5 IN;
PAGE LENGTH = 11 IN;
```

or

```
PAPER = LETTER;
```

LETTER is one of the paper size options in TPL REPORT. For other built-in paper sizes, see the FORMAT statement called **PAPER**.

For most printers, PostScript printing requires a margin. If you try to print something that fills the paper to the edges, you may lose part if it. Therefore, we do not recommend margin sizes of 0 when using PostScript.

Switching between Line Printer and PostScript Modes

In general, if you will be switching between line printer and PostScript mode, sizes other than those for the page and its margins will work well in both modes if they are expressed in characters. If you are using a proportional font in PostScript, you will often be able to get more characters within a given width. The most common exception is when you have a label in upper case letters. Upper case letters are often wider in a proportional font.

Sizes specified in inches, centimeters or points will work in line printer mode as well as in PostScript mode. If you are not requesting PostScript output, the measures will be converted to 12 pt equivalents in characters. With 12 pt type, 1 inch can contain 10 characters in the horizontal direction and 6 lines in the vertical direction.

Jobs that have been set up for PostScript printing can be converted to ascii text mode by simply changing the FORMAT specification to **POSTSCRIPT = NO;.** Any other FORMAT specifications that relate only to PostScript will be ignored. Again, you should be aware of size specifications. For example, a column width, such as .7 in, that is appropriate for a font with a small size may seem very narrow when you convert to ascii text mode.

Report Output Files

Ascii text reports are saved with file names such as **a1.rep** or **industry.rep.** The comparable PostScript reports are saved in files named **a1.ps** or **industry.ps**.

UNIX Note Since PostScript reports cannot be reviewed on the screen, you may wish to prepare your reports in "draft" form with POSTSCRIPT set to **NO**, then switch to **POSTSCRIPT = YES;** when you know that the report content is correct. If you wish, you can transfer PostScript reports to a Windows PC to review them using View or Ted, the TPL Editor. Please contact QQQ Software, Inc. if you need one of these programs.

Printer Selection -- UNIX

If you have both a PostScript and a non-PostScript printer attached to your computer, you can use a FORMAT statement to choose the appropriate printer. See the **PRINT COMMAND** statement in the FORMAT chapter for details.

Using PostScript Reports with other Software

If you ask TPL REPORT to prepare the PostScript reports for use by other programs, it will divide the reports into pages and save each page in a separate Encapsulated PostScript (EPS) file. Each file will have a name that is a combination of the the **page number**, the **report number** and the suffix **.eps**. Assume a report request with 3 reports, where the first report has two pages and the other reports have a single page. If the report names are A1, A2 and A3, the EPS files will be named:

```
P1R1.EPS
P2R1.EPS
P3R2.EPS
P4R3.EPS
```

This process is described in detail in the appendix "Run Instructions". If you have many pages, refer to the **PAGE MARKER** statement in the FORMAT chapter for a good way of marking the pages with the file names.

Font Selection

Fonts can be selected for report elements, such as titles, variable labels or masks, with FORMAT **FONT** statements. Font specifications can also be included in individual labels and masks for additional customizing. These detailed specifications are described in the chapters on labels and masks.

PostScript Examples

Following are FORMAT statements requesting PostScript output and the report that was formatted using these statements. Note that if we change the PostScript statement to **POSTSCRIPT = NO;**, the **FONT** and **DOWN RULE WEIGHT** statements will be ignored, because they only affect PostScript reports.

```
POSTSCRIPT = YES;
PAPER = LETTER;

DEFAULT FONT = H 10;          /* Helvetica 10 */
TITLE FONT = HB 12;  /* Helvetica-Bold 12 */
VARIABLE LABEL FONT TI 12;    /* Times-Italic 12 */

RETAIN ALL RULES;
RETAIN SIDE RULES;
FOR VARIABLE NUMBER:  DOWN RULE WEIGHT = 2;

FOR COLUMN 1:         ALIGN COLUMN LEFT;
                      COLUMN WIDTH 20;
```

Employee Report for December 1998

Row	*Type of Job*	*Pay Rate*	*Sex*
1	Clerk/Typist	5.53	f
2	Director of Operations	21.49	m
3	Staff Secretary	7.35	f
4	Executive Secretary	9.21	f
5	Accounting Clerk	5.00	m
6	Payroll Clerk	5.19	f
7	Accounting Supervisor	8.65	f

Dashes in PostScript

There are three sizes of "dash" characters available in PostScript. A short dash is used for hyphenation. This dash is the hyphen character on your keyboard. A medium dash is used as the footnote symbol for the EMPTY built-in footnote ("Data not available."), and a long dash is used in title continuations. The choice of dash can be changed for the title continuation with the FORMAT statement **REPLACE TITLE CONTINUATION;** .

The medium and long dashes are special characters that are not on your keyboard but can be entered with \nnn where nnn is a character code. The choice of character code depends on the TPL Codepage being used. See the appendix called "Character Sets" for details about Codepages. A better way to enter these characters is by using their names with **&** in front and **;** following. **&endash;** can be used to specify the medium dash while **&emdash;** can be used for the long dash.

Suppose we wish to replace the dash in the title continuation with an endash. We could use the statement:

Example

```
REPLACE TITLE CONTINUATION WITH '&endash; continued';
```

CHAPTER 18

Color and Grey

USING COLOR IN REPORTS

General Information on Color

You can specify color for individual labels, including titles, and for masks within codebooks, report requests and FORMAT requests. Color defaults for data values, labels and rules (horizontal and vertical lines) can be specified with FORMAT statements.

The word **COLOR** can also be spelled **COLOUR**.

COLOR specifications are only effective in PostScript mode. They are ignored in line printer mode.

Effect on Monochrome Printers

If reports that use color are printed on a non-color PostScript printer, the colors will print as shades of grey. See the FORMAT statement COLOR = NO for tips on switching between color and monochrome.

Important This chapter is best viewed in the pdf form since colors are not displayed in the paper manual.

r g b colors

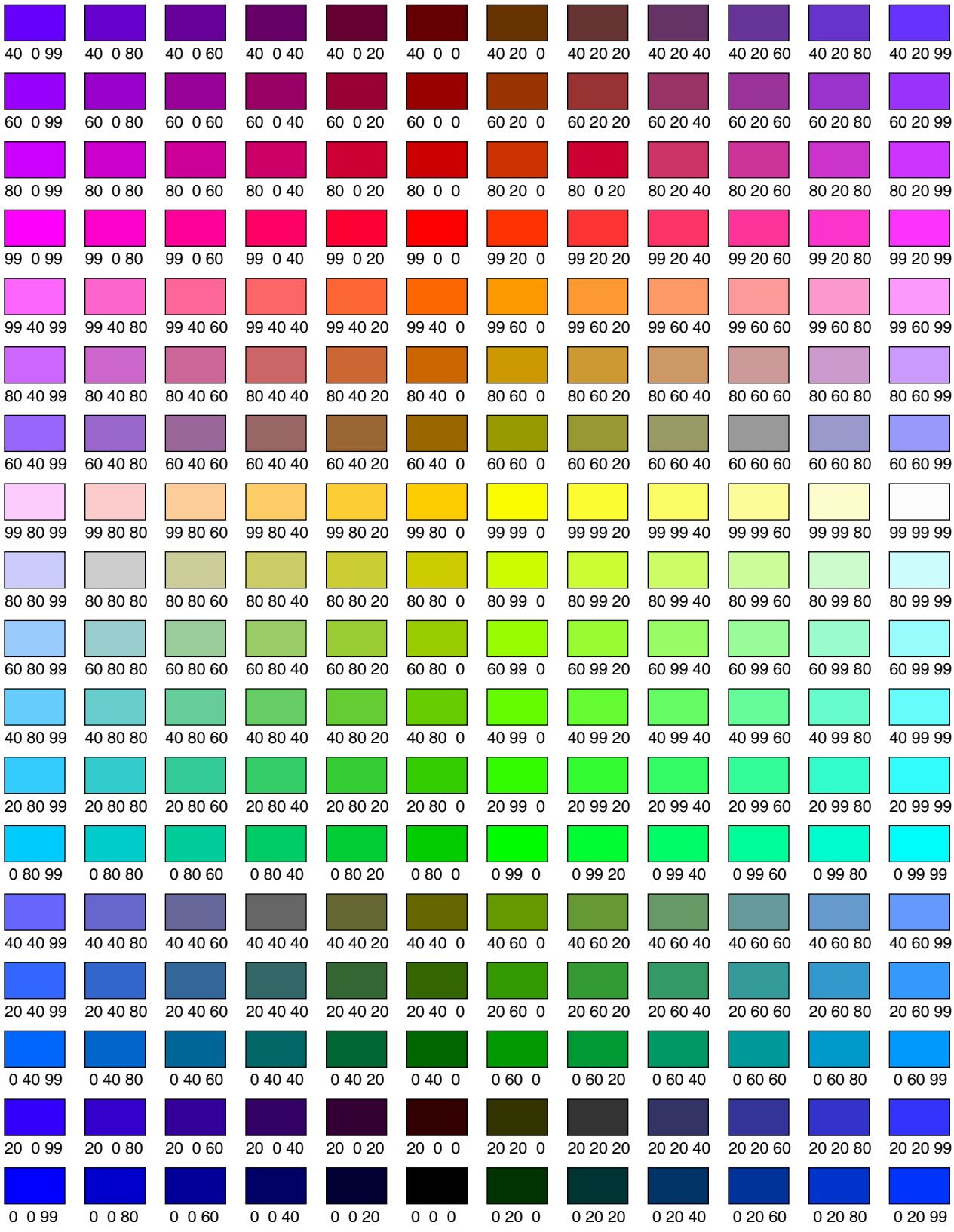
Colors are specified by a combination of red, green and blue. We will refer to them as r g b where the amount of each color in the mix is indicated by a value from 0 to 100. For example, the color blue has the r g b numbers 0 0 100. In other words, there is no red, no green, and the maximum amount of blue.

Color Chart

A color chart file called **colors.ps** is included with the TPL system. It can be found in either the **doc** subdirectory or the **examples** subdirectory of the TPL system directory, depending on the version of TPL you have. If you have a color PostScript printer, you can print this chart on your color printer to see what colors are printed for a variety of r g b colors.

The color chart is also shown on the next page of this manual. With the PDF version of the manual open in Adobe Acrobat Reader, you can print the page on your color printer.

There is very little consistency between color printers, so the same r g b color printed on one color printer may look quite different when printed on another. With the color chart, you can choose precisely from the colors as printed by your printer


40 0 99
40 0 80
40 0 60
40 0 40
40 0 20
40 0 0
40 20 0
40 20 20
40 20 40
40 20 60
40 20 80
40 20 99
60 0 99
60 0 80
60 0 60
60 0 40
60 0 20
60 0 0
60 20 0
60 20 20
60 20 40
60 20 60
60 20 80
60 20 99
80 0 99
80 0 80
80 0 60
80 0 40
80 0 20
80 0 0
80 20 0
80 0 20
80 20 40
80 20 60
80 20 80
80 20 99
99 0 99
99 0 80
99 0 60
99 0 40
99 0 20
99 0 0
99 20 0
99 20 20
99 20 40
99 20 60
99 20 80
99 20 99
99 40 99
99 40 80
99 40 60
99 40 40
99 40 20
99 40 0
99 60 0
99 60 20
99 60 40
99 60 60
99 60 80
99 60 99
80 40 99
80 40 80
80 40 60
80 40 40
80 40 20
80 40 0
80 60 0
80 60 20
80 60 40
80 60 60
80 60 80
80 60 99
60 40 99
60 40 80
60 40 60
60 40 40
60 40 20
60 40 0
60 60 0
60 60 20
60 60 40
60 60 60
60 60 80
60 60 99
99 80 99
99 80 80
99 80 60
99 80 40
99 80 20
99 80 0
99 99 0
99 99 20
99 99 40
99 99 60
99 99 80
99 99 99
80 80 99
80 80 80
80 80 60
80 80 40
80 80 20
80 80 0
80 99 0
80 99 20
80 99 40
80 99 60
80 99 80
80 99 99
60 80 99
60 80 80
60 80 60
60 80 40
60 80 20
60 80 0
60 99 0
60 99 20
60 99 40
60 99 60
60 99 80
60 99 99
40 80 99
40 80 80
40 80 60
40 80 40
40 80 20
40 80 0
40 99 0
40 99 20
40 99 40
40 99 60
40 99 80
40 99 99
20 80 99
20 80 80
20 80 60
20 80 40
20 80 20
20 80 0
20 99 0
20 99 20
20 99 40
20 99 60
20 99 80
20 99 99
0 80 99
0 80 80
0 80 60
0 80 40
0 80 20
0 80 0
0 99 0
0 99 20
0 99 40
0 99 60
0 99 80
0 99 99
40 40 99
40 40 80
40 40 60
40 40 40
40 40 20
40 40 0
40 60 0
40 60 20
40 60 40
40 60 60
40 60 80
40 60 99
20 40 99
20 40 80
20 40 60
20 40 40
20 40 20
20 40 0
20 60 0
20 60 20
20 60 40
20 60 60
20 60 80
20 60 99
0 40 99
0 40 80
0 40 60
0 40 40
0 40 20
0 40 0
0 60 0
0 60 20
0 60 40
0 60 60
0 60 80
0 60 99
20 0 99
20 0 80
20 0 60
20 0 40
20 0 20
20 0 0
20 20 0
20 20 20
20 20 40
20 20 60
20 20 80
20 20 99
0 0 99
0 0 80
0 0 60
0 0 40
0 0 20
0 0 0
0 20 0
0 20 20
0 20 40
0 20 60
0 20 80
0 20 99

Color Definitions in color.tpl

Colors can also be referenced by name where the colors have been defined in a file called **color.tpl**. Establishing color definitions in this way can be very convenient if you have a set of standard colors that you use frequently, because you do not need to remember the r g b values for the colors. Instead, you can reference the colors by name. This approach also allows you to choose different sets of standard color definitions for different printers and adjust your color output to the different printers simply by using a different color.tpl file.

The TPL REPORT installation process creates a file called color.tpl and puts it in the TPL system directory. Several examples of color definitions are included in this file. You can edit it to change or add to the color definitions. If you want to leave the system color file unchanged but use a different set of color definitions for your own jobs, you can make a copy of color.tpl in the subdirectory where you are working and edit it to include your own set of color definitions. The color definitions in the directory where you are working will override the ones in the color.tpl file in the TPL system directory.

The format of a color definition in color.tpl is:

Format

```
color r g b
```

where color is a name that you choose to associate with a specific color definition and r, g and b are numbers between 0 and 100 (inclusive) which specify the **r**ed, **g**reen, and **b**lue components of a color.

Note Color definitions entered in color.tpl DO NOT end with a semicolon (;).

Note If you enter a color definition in color.tpl with the color name GRAY or GREY, it will be ignored. These names are reserved for grey characters and shading.

Example Following is an example of a color.tpl file:

```
red            100    0    0
green            0  100    0
blue             0    0  100
brown           60   40    0
cyan             0  100  100
yellow         100  100    0
light_yellow   100  100   20
purple          40    0  100
magenta        100    0  100
orange          90   60    0
black            0    0    0
```

Effect The colors red, green, blue, brown, cyan, yellow, light_yellow, purple, magenta, orange and black are defined in the color.tpl file and can be referenced by name in any TPL REPORT color specifications.

Example To choose the color RED as the default for all characters and rules in a report, you can use the FORMAT statement:

```
DEFAULT COLOR = RED;
```

This statement has the same meaning as the statement:

```
DEFAULT COLOR = 100 0 0;
```

Example To choose the color BLUE for the report title and column labels, you can use the FORMAT statement:

```
LABEL COLOR = BLUE;
```

This statement has the same meaning as the statement:

```
LABEL COLOR = 0 0 100;
```

Employee Report for December 1998

Row	*Type of Job*	*Pay Rate*	*Sex*
1	Clerk/Typist	5.53	f
2	Director of Operations	21.49	m
3	Staff Secretary	7.35	f
4	Executive Secretary	9.21	f
5	Accounting Clerk	5.00	m
6	Payroll Clerk	5.19	f
7	Accounting Supervisor	8.65	f

Note on Changing Color Definitions in color.tpl

If you include color names in individual labels, including titles, or in masks within codebooks or report requests, you should be aware that these colors are "built into" the labels and masks when the codebooks or report requests are first run. TPL REPORT converts the color names to the literal r g b numbering and saves this numbering as part of the labels or masks as they are processed. Thus, if a codebook is processed with one set of color name specifications and then the color.tpl file is changed, the old color specifications will continue to apply to the labels in that codebook until the codebook is reprocessed. Similarly, changing color.tpl before a TPL REPORT rerun will be effective for FORMAT statement colors but not for report request colors.

Recommendation

If you reference colors by name in a codebook, then change the color.tpl file, you will probably want to reprocess the codebook to switch to the new color definitions.

If you run a report request in which colors are referenced by name, then change color.tpl, you will probably want to run the job over from the beginning to get the new color definitions. Doing a rerun with FORMAT statements will not change the original label or mask colors that were assigned in the report request.

Printing Color Separations for Reports

Color separations cannot be printed directly from TPL REPORT, but you can print them easily using desktop publishing software. First, ask TPL REPORT to convert your reports to Encapsulated PostScript (EPS) files. The reports will be converted into EPS files with one report page per file. You can then bring the resulting EPS report pages into documents created with desktop publishing software. If the reports have color, the EPS files will automatically include the information needed for the desktop publishing software to print CMYK color separations. You do not need to do anything special to make this happen.

Example In PageMaker, after bringing an EPS report page into the document, choose "Print", then "Color", then click on "Separations".

The Special Color GREY

You can specify GREY in any situation where COLOR is allowed. GREY prints equally well on both color and non-color PostScript printers. It can be particularly useful for shading if you have a non-color PostScript printer. It is less useful for labels or data values, since letters and numbers do not print very well in a grey shade. GREY is specified with a number between 0 (white) and 100 (black).

*******NOTE: Shading has not been implemented for TPL REPORT*******

GREY can also be spelled **GRAY**.

The following example show the use of GREY in FORMAT statements.

Examples

```
REPLACE LABEL WITH GREY 30 'A grey label';
```

Since GREY is a special built-in color, the color names GREY and GRAY can only be used with a number that specifies the degree of shading. If there is a definition of GREY or GRAY in the color.tpl file, the definition will be ignored.

Color Specifications for Individual Labels and Masks

Color can be added to labels, including titles, and to masks within codebooks, report requests and format requests. Color specified in individual labels and masks overrides colors specified in the FORMAT statements, DEFAULT COLOR and LABEL COLOR.

Labels

COLOR can be used freely within a label in the same way as other types of label characteristics such as fonts, spacing and line breaks.

Example

```
"This is" COLOR RED " a two-tone label." COLOR 20 20 80
```

The label will print as follows:

"This is" will be printed in the default label color.

" a two-tone label." will be printed in RED as defined in the color.tpl file.

Masks

COLOR can appear anywhere in a data mask. Note that this is different from a FONT specification which must be at the end of a mask. When used in a data mask, COLOR applies to the entire mask.

Example

```
MASK  COLOR 100 0 0  999.99 '%'
```

The data values and percent sign will be printed in color 100 0 0 (red).

RECODE Values

You can assign colors to new values entered in a RECODE statement whenever these values contain label elements.

Setting COLOR Defaults for Characters and Rules

COLOR defaults can be set for all characters and rules used in reports. Defaults can be set in either the profile or a format request using the following FORMAT statements. Colors can be specified in r g b format or using color names that have been defined in color.tpl.

Format

```
DEFAULT COLOR = color;
LABEL COLOR = color;
```

```
RULE COLOR = color;
SYMBOL COLOR = color;
```

Color defaults are applied as described below. For additional details, see the section on "COLOR Defaults" in the FORMAT chapter. In cases where color specifications are entered directly into ***individual*** report elements such as labels, masks or footnotes, these individual specifications will take precedence over the ***default*** COLOR specifications.

DEFAULT COLOR is the print color for the entire report if no other colors are specified. If you do not set DEFAULT COLOR, the default is black. If RULE COLOR and LABEL COLOR are specified, the DEFAULT COLOR remains as the default color for report cells.

RULE COLOR is the print color for rules. It applies to all rules, including rules added by the FORMAT statement RULE EACH. If no explicit RULE COLOR is specified, rules are printed in the default color.

LABEL COLOR is the print color for all text in reports except character strings in cell masks. These strings are printed in the default color. If no explicit LABEL COLOR is specified, all labels and titles are printed in the default color.

SYMBOL COLOR is the print color for all footnote symbols. If SYMBOL COLOR is not set explicitly, the default label color is used for symbols.

Example

```
DEFAULT COLOR = 0 20 99;
FOR REPORT 1:  REPLACE TITLE WITH COLOR RED 'Red report title';
FOR REPORT 2:  RULE COLOR = RED;
```

Effect

All reports will be printed in the default color 0 20 99 (a shade of blue) except as follows. The first report will have a red title. The rules in the second report will be red, and the rest of the second report will be printed in the default color.

Replacing Mask Color

With the FORMAT statement, REPLACE MASK COLOR, you can replace the color of a mask without disturbing any other specifications in the mask and without re-entering the entire mask. *Unlike most MASK statements, this one applies to all types of variables, observation, control and char.*. See the FORMAT chapter for complete details.

Mask color can be replaced for a single cell, a group of cells or the entire report. If you replace the mask color for an entire report, the mask color serves as a de-

fault color setting for the report cells without affecting other parts of the report that are colored by the DEFAULT COLOR statement.

Format

```
REPLACE MASK COLOR WITH  color;
```

The **color** can be specifed in **r g b** format or using color names that have been defined in **color.tpl**.

Example

```
FOR ROW 1:  REPLACE MASK COLOR WITH RED;
FOR ROW 1 COLUMN 1:  REPLACE MASK COLOR WITH BLUE;
FOR VARIABLE INCOME:  REPLACE MASK COLOR WITH GREEN;
```

Effect

The mask color for the first row will be red except in column 1 where the mask color will be blue. The rows and/or columns containing INCOME values will have a mask color of green.

CHAPTER 19

Exports

CONVERTING POSTSCRIPT REPORTS TO OTHER FORMATS

Introduction

Reports created in PostScript mode can be exported to CSV (delimited) or EPS files. This chapter describes the export of CSV files. EPS (**E**ncapsulated **P**ost-**S**cript) files are described elsewhere in the manual.

PDF Format

Adobe Acrobat PDF format is a commonly used format for exchanging documents including reports. It is often used for web display when the author wishes to display the document with more precise formatting than is available using HTML. The Acrobat Reader, which is used for displaying the PDF file, is available free.

PostScript reports can be distilled to Acrobat PDF format. In the Windows version of TPL REPORT, you can do this as an export from TED if you have Adobe Acrobat Distiller available on your computer. If you do not have Acrobat Distiller, you can install and use Ghostscript (gs815w32.exe) to create PDF files. After TPL REPORT is installed, an installation icon for GPL Ghostscript may be on your desktop. Otherwise, it can be found in C:\program files\gpl. See ***PDF*** in TED Help for more information.

CSV (delimited) Export

In exported CSV files, each cell value is contained in double quotes and the values are separated by commas. The wafer labels, if any, and the stub labels are added to the data as extra columns at the beginning of each row. If you do not want these label columns, delete the wafer labels and the stub before exporting. The

bottom level of the heading is used as the first row of the CSV file. This row provides field names for the columns of the file. If you do not want this row of names, delete the heading before exporting. You may also change the heading labels before exporting if you want better field names.

Footnotes symbols are not included in the labels or values, but other mask items, such as $, %, or mask text, are retained in the data values.

See the Format statement CSV DIVIDER to separate the values with a character other than comma.

Windows Note If you are exporting interactively from TED, there is an option to enter a character other than comma to be used as the divider between the values in the exported file(s), or you can select Tab as the divider.

CSV Files

By default, exported CSV files are saved in a job's **TPLRnnnn** subdirectory. One CSV file is created for each report in the job with a name that includes the **report-name** and a suffix of **.csv** on the file name. For example, if the report is **CPS_REP** then the csv file will be **cps_rep.csv**.

Windows Note If you are exporting interactively from TED, the exported file(s) will be saved using the **File Name** and **Current Directory** shown in the Export screen. You can change the name and directory if you wish.

How to Request CSV Export

Only PostScript tables can be exported to CSV files. Instructions depend on whether you are running TPL REPORT under Windows or UNIX.

Windows

CSV files are exported from TED. See *TED Help* for details on exporting interactively. To produce CSV in a batch job, see *TPL Help* or the appendix on Scripts.

UNIX

If you run TPL using the command line prompts, a prompt will ask if you want to export CSV. To produce CSV using a command argument, see the appendix with UNIX Run Instructions. See also the Format statement CSV OUTPUT = YES/NO to prevent the prompts.

CHAPTER 20

TPL-SQL

INTRODUCTION TO THE DATABASE INTERFACE

TPL-SQL is an optional database interface for TPL. It allows TPL Tables and TPL Report to read data directly from a SQL database. When you use the interface, you do not need to first extract the data from the database. So you do not need space to store the extracted data. You also do not need to know SQL. TPL automatically generates the request to extract just the data it needs. It processes the data as it extracts it, so there is not even a need for extra temporary storage. Further, TPL does not write on your database. Anyone with read access to the data can produce tables or reports.

In TPL Tables and TPL Report, there is very little difference between accessing a stand-alone sequential file and accessing one or more relations stored in a database. If you already know TPL Tables or TPL Report and know the structure of your data, you will find it very easy to use TPL-SQL. The primary differences between processing a sequential file and processing a database are found in describing the data in your codebook.

ODBC Note This chapter applies to all versions of TPL-SQL, including the Windows version that accesses databases via ODBC. ***If you are using the Windows version of TPL-SQL, we recommend that you use Codebook Builder to generate a codebook.*** Most of the information contained in this chapter is also included in the Codebook Builder Help, along with instructions on how to use Codebook Builder.

TERMINOLOGY - YES, YOU WANT TO READ THIS

Unfortunately, TPL Tables and relational theory use the same words to mean different things. In relational terminology, a *table* is approximately the equivalent of a flat sequential data file in TPL terminology. In TPL terminology, a *table* is the final product of a TPL Tables run. To avoid confusion, when we refer to a *table*

we mean the output of a TPL Tables run. We will refer to the data file used in relational terminology as a *SQL table* or a *relation*. *Variable* and *field* are used interchangeably. The SQL term *column* will not be used as this could be confused with TPL Tables and TPL Report output columns.

TPL-SQL CODEBOOK

The primary difference between using TPL Tables or TPL Report with a sequential file and using them with TPL-SQL can be found in the description of the data contained in the TPL codebook.

Note Codebooks for SQL databases cannot contain repeating groups.

Sequential files may be either flat files or hierarchical files. Each row of a sequential file is called a *record*. If all records are of the same format, the file is flat. If records of different format are interspersed, the file is called *hierarchical*. The order of the records on the sequential file determines the hierarchical membership; i.e. the children immediately follow their parent.

When using TPL-SQL, TPL Tables and TPL Report process data stored in SQL databases as hierarchies. But in the case of a SQL database, records of different types are not interspersed. Instead, the records of each type are stored in separate SQL tables. Since the order of records cannot be used to describe hierarchical relationships, something else must be used. This other thing is a pairing between fields on different SQL tables. We call such a pairing, an *association*, and statements which define them are *association statements*. The fields in association statements are frequently but not always *key* fields. They are the fields that are specified in SQL requests in the *where* clause of *joins*.

A TPL codebook describing a sequential hierarchical file consists of a description of one or more records. Each record description consists of a description of its constituent fields. A TPL codebook which uses TPL-SQL consists of descriptions of one or more SQL tables. Each SQL table description consists of a description of its constituent fields. In addition, the TPL-SQL codebook includes association statements which define how multiple SQL tables are to be processed together.

A Simple TPL-SQL Codebook Example

Though much of this document discusses hierarchical files, we will begin with a flat file example. Suppose we have a flat sequential file describing a family. The TPL codebook might be:

Sequential File Version

```
Begin Families Codebook
Family  Record
Filler 7
Region control 1
        (
                "Northeast"      = 1
                "North Central" = 2
                "South" = 3
                "West"  = 4
        )
Living_Qrt  "Living Quarters" control 1
        (
                "Owned"         = 1
                "Condominium" = 2
                "Rented"        = 3
                "Unknown"       = " "
        )
Persons_in_family obs 2
Gross_income_of_head obs 7
Gross_income_of_spouse obs 7
End Families Codebook
```

If we now load our data into a SQL database we can describe our data file to TPL using the following:

SQL Database Version

```
Begin Families Codebook SQL
Family defines "family" Table
Region defines "region" control 1
        (
                "Northeast"      = 1
                "North Central" = 2
                "South" = 3
                "West"  = 4
        )
Living_Qrt  defines "living_qrt" "Living Quarters" control 1
        (
                "Owned"         = 1
                "Condominium" = 2
                "Rented"        = 3
                "Unknown"       = " "
        )
Persons_in_family defines "persons_in_family" obs 2
Gross_income_of_head defines "gross_income_of_head" obs 7
Gross_income_of_spouse defines "gross_income_of_spouse" obs 7
End Families Codebook
```

The only necessary changes are that **SQL** is added after **Codebook**; **Record** becomes **Table**; and the **Filler** field is eliminated. In most cases a **defines** clause is also needed. A change in meaning which is not discernible concerns the order of fields in the record or SQL table description. In a sequential file description, the fields must be listed in the order they occur in the file. In a SQL database description, the order of the fields is arbitrary within a SQL table. Another change is that it is not necessary to describe all of the fields in a SQL table in your codebook. Perhaps some of the fields are confidential and should not be used. Other fields may be text fields which are not appropriate for tables. If you do not have TPL Report, you may wish to omit these fields. For sequential file codebooks you are required to mark the space these fields occupied as **Filler**. In a codebook describing a SQL table, the fields can just be omitted.

Defines Clause

The **defines** clause is used to map TPL variable names into SQL database field names. If the **defines** clause is omitted, TPL assumes that the SQL database field name is the same as the TPL variable name except that it is all uppercase. In the above example, all of the SQL database field names are assumed to be in lowercase. So the **defines** clauses are necessary. Define clauses are discussed more fully later.

A Better Solution - Using Information from the Database

The simple transformation above is not the recommended way of creating a TPL-SQL codebook. The SQL database contains much of the information contained in the TPL codebook. If the TPL codebook information does not match the database information, then errors or incorrect reports will result. Thus instead of requiring the transformation described above, TPL provides a way to automatically query the database for the relevant information contained within the database. You activate this processing by omitting the information you can obtain from the database. The recommended codebook source is:

You Write

```
Begin Families Codebook SQL
Family Table
Region defines "region" control  get conditions from data
Living_Qrt  defines "living_qrt" "Living Quarters" control from data
Persons_in_family defines "persons_in_family" obs
Gross_income_of_head defines "gross_income_of_head" obs
Gross_income_of_spouse defines "gross_income_of_spouse" obs
End Families Codebook
```

In this codebook source, field widths and obs modifiers such as **float** have been eliminated as have control variable condition values lists. Instead of listing the condition values, we have included **get conditions from data** or its shorter synonym **from data**.

Unix

On computers running Unix, use the **tpl conditions** program to process the above codebook source. The **tpl conditions** program will fill in the field widths, datatype details and control variable conditions to create a new codebook source. The new codebook source shown below can be edited to provide better condition names. It should then be run through **tpl codebook** to produce a compiled codebook. For more details, see the Appendix titled ***TPL Conditions*** and the sections on **tpl conditions** and **tpl codebook** in ***Run Instructions (UNIX)*** for more details.

TPL-SQL Generates

```
Begin Families Codebook SQL
Family defines "family" Table
Region defines "region" control 1
        (
                "1"     = "1"
                "2"     = "2"
                "3"     = "3"
                "4"     = "4"
        )
Living_Qrt  defines "living_qrt" "Living Quarters"  control 1
        (
                "1"     = "1"
                "2"     = "2"
                "3"     = "3"
                " "     = " "
        )
Persons_in_family  defines "persons_in_family"  obs 2
Gross_income_of_head defines "gross income of head"  obs 7
Gross_income_of_spouse defines "gross_income_of_spouse"  obs 7
End Families Codebook
```

Windows

On computers running Microsoft Windows, the above codebook source is processed directly by the codebook processor to produce the following source. The data source and password if required are also provided to the **codebook** program. See ***Run Instructions (Windows)*** and ***Scripts (Windows)*** for more details.

NOTE: It is important for Windows sources that each variable description start on a new line. Otherwise the new source described below may not be correct.

When the codebook processor has completed its work, a new codebook source is created with **evaluated to** expressions to show the information obtained from the database. The actual conversion depends upon both the data and the database management system. A possible conversion of our example follows:

TPL-SQL Generates

```
Begin Families Codebook SQL
Family defines "family" Table
Region defines "region" control get conditions from data evaluated to control 1
        (
                 "1"     = 1
                 "2"     = 2
                 "3"     = 3
                 "4"     = 4
        )
Living_Qrt  defines "living_qrt" "Living Quarters" control from data
        evaluated to control 1
        (
                "1"      = 1
                "2"      = 2
                "3"      = 3
                " "      = " "
        )
Persons_in_family  defines "persons_in_family" obs evaluated to obs 2
Gross_income_of_head defines "gross income of head" obs
        evaluated to obs 7
Gross_income_of_spouse defines "gross_income_of_spouse" obs
        evaluated to obs 7
End Families Codebook
```

You may wish to edit and reprocess this new source. This will enable you to do such things as provide better labels for condition values. If the codebook name is *Families*, the new codebook source will be *FAMILIES.S*.

Conversions from Database to TPL Data Types

In our example, we assumed that all of the data was loaded into the database as character fields of the same length as the original data. This is not necessary. SQL databases support many formats of data and conversions of data from one format to another. However, you as a user of TPL do not need to worry about these conversions. In general all you need to do is label the fields as **observation**, **control**, or **character**. The system will correctly determine the exact data type and place it in the **evaluated to** clause. You can use these data types explicity, but this is not recommended.

Restrictions There are some restraints on what fields you can label **observation** or **control**. For example, if a data field contains alphabetic characters, it should not be used as an observation. TPL will detect and report some of these errors when the codebook is created. Others will only be detected when table or report jobs are processed.

ODBC Data Type Conversions

The following chart shows the acceptable conversions from ODBC to TPL datatypes.

		TPL Types		
ODBC Types		**Obs**	**Con**	**Char[2]**
date		y	y	y
time		y	y	y
timestamp		y	y	y
char		y[1]	y	y
varchar		y[1]	y	y
longvarchar		y[1]	y	y
numeric	y	y	y	
decimal		y	y	y
tinyint		y	y	y
smallint		y	y	y
integer		y	y	y
bigint		y	y	y
float		y	y	y
double		y	y	y
binary		y	y	y
varbinary		y	y	y
longvarbinary		y	y	y
bit		y	y	y
real		y	y	y

[1] If a character string described as obs contains non-numeric values, this will be detected when a table or report is processed. It will not be detected by the codebook processor.

[2] **Character** may be a new data category for you. It is similar to control but without the requirement of a list of possible conditions. In TPL Report it can be used in selects, recodes, conditional computes or report statements. In TPL Tables it can be used in selects, defines or conditional computes but not directly in tables

Oracle Data Type Conversions

The following chart shows the acceptable conversions from Oracle to TPL data types.

	TPL Types		
Oracle Types	**Obs**	**Con**	**Char[2]**
date	y	y	y
char	y[1]	y	y
varchar[2]	y[1]	y	y
varchar	y[1]	y	y
number	y	y	y
long	n	n	n
raw	n	n	n
long raw	n	n	n
rowid	n	n	n
mslabel	n	n	n

[1] If a character string described as obs contains non-numeric values, this will be detected when a table or report is processed. It will not be detected by the codebook processor.

[2] **Character** may be a new data category for you. It is similar to control but without the requirement of a list of possible conditions. In TPL Report it can be used in selects, recodes, conditional computes or report statements. In TPL Tables it can be used in selects, defines or conditional computes but not directly in tables

Sybase Data Type Conversions

The following chart shows the acceptable conversions from Sybase to TPL datatypes.

	TPL Types		
Sybase Types	**Obs**	**Con**	**Char[2]**
char	y[1]	y	y
varchar	y[1]	y	y
bit	y	y	y
binary	y	y	y
tinyint	y	y	y
smallint	y	y	y
integer	y	y	y
float	y	n	n
long float	y	n	n
money	y	n	n
datetime	y	y	y
decimal	y	y	y
numeric	y	y	y
image	n	n	n

[1] If a character string described as obs contains non-numeric values, this will be detected when a table or report is processed. It will not be detected by the codebook processor.

[2] **Character** may be a new data category for you. It is similar to control but without the requirement of a list of possible conditions. In TPL Report it can be used in selects, recodes, conditional computes or report statements. In TPL Tables it can be used in selects, defines or conditional computes but not directly in tables

New Data Types

In addition to the usual codebook data types, the following data types may be generated in the **evaluated to** clause. You should not enter these directly. However, you may wish to add a *time-unit* to generated **obs date** fields.

obs varying and **con varying** — These are just regular observation and control variables stored on the database as varying length fields. TPL Tables and TPL Report automatically handle these data types so you can treat them as if they are normal fixed-length fields. "Short" control variable values are right-padded with blanks. This data type is currently implemented for databases only.

character varying — TPL Report does not pad these fields when they are displayed. This data type is currently implemented for databases only.

money or **obs money** — This is a floating point data type. If there is no explicit mask provided, the system defaults to a mask of **$999.99**. This data type is currently implemented for databases only.

control date or **character date** — The format of the date is determined by a database environment variable. If you change the value of this environment variable, you must rerun your codebook against the database before you run a table or report request. Dates are sorted and displayed in chronological order rather than character sort order. This data type is currently implemented for databases only.

Sybase If client Sybase software is not installed on the computer on which TPL is running and installation Option 1 was used to install TPL, **control date** fields will be displayed in US English. If you wish to display dates in a different format, then you can explicitly change the generated labels.

obs date *time-unit* — where *time-unit* may be **year**, **day**, **hour**, **minute**, or **second**. If you chose to explicitly call a field **obs date** and omit *time-unit*, then the system will assume a *time-unit* of **day**. This data type is currently implemented for databases only.

Oracle The field evaluates to the number of units since January 1, 1900 at 00:00:00. Dates are floating point numbers so decimal values will be shown if a mask is provided which specifies them.

Sybase The field evaluates to the number of units since the current time (when job is run). Thus all past dates are negative and future dates are positive. Dates are truncated integer values not decimal fractions.

ODBC The meaning of time units in databases accessed via ODBC depends upon the underlying database system. Consult your database manuals for information on this.

The **obs date** data type is especially useful for computing time intervals; e.g.,

```
Compute Life_span = Death_date_in_years - Birth_date_in years;
```

Note that the time unit for the two terms in the compute must be the same or a trash answer will result.

Label-Code Tables

It is common in SQL databases to have SQL tables which pair code values with longer descriptions or labels. This can save a considerable amount of space in your database. It also allows the description to be changed without changing large numbers of database records. When an extract is made from the database, a SQL join is normally performed so that the label rather than the code is displayed with the other data fields. The TPL codebook can make use of these label-code pairs to create condition sets for control variables.

In our earlier example of codebook generation we included the line:

```
Living_Qrt  "Living Quarters" control from data
```

This generated the following:

```
Living_Qrt  "Living Quarters" control from data evaluated to control 1
        (
                "1"     = 1
                "2"     = 2
                "3"     = 3
                " "     = " "
        )
```

Suppose we had in our database a SQL table **lq_tab** with fields **lq_code** and **lq_lab** and values:

lq_code	lq_lab
1	Owned
2	Condominium
3	Rented
' '	Unknown

The **lq_code** field in the **lq_tab** SQL table has the same range of values as the **Living_Qrt** field of the **family** SQL table. So we can substitute the new TPL codebook statement:

```
Living_Qrt "Living Quarters" control from lq_tab(lq_lab,lq_code)
```

The result is:

```
Living_Qrt  "Living Quarters" control from lq_tab(lq_lab,lq_code)
evaluated to control 1
        (
                "Owned"         = 1
                "Condominium" = 2
                "Rented"        = 3
                " Unknown"      = " "
        )
```

The SQL table containing the label-code pairs can be referenced in this way without itself being described elsewhere in the codebook.

The label and code fields must be put in the parentheses in the order shown above, that is *label first* and *code second*.

Alternate Names - The DEFINES Clause

By default, the TPL name for a field is the same as the name of the field on the database. There are some cases where this is not desirable. For example, in a sequential file we sometimes wish to use the same field as both an observation variable and a control variable. We accomplish this by using a **redefine** clause. TPL-SQL codebooks cannot have **redefine** clauses, but the **defines** clause can be used to accomplish the same result.

```
tpl-name1 defines sql-name  control from data
tpl-name2 defines sql-name obs
```

The *sql-name* is the name for the database field. It may be placed within quotation marks. This is useful if it happens to be a TPL keyword or is otherwise not a valid name for a TPL variable. *tpl-name1* and *tpl-name2* are two TPL variable names. They can be used in table and report requests as well as association statements in the codebook. The only place the SQL name can be used is in the special SQL SELECT statements discussed later.

A codebook description with a **defines** may include all of the standard codebook field qualifiers; e.g.,

```
Living_quarters  "Living Quarters" defines Living_Qrt control
condition label is "Housing type = " value  from data
```

Sybase Sybase is case sensitive; that is, lower-case letters and upper-case letters are not treated as equal. TPL is case sensitive only for items within quotes. Thus if Sybase fields were given lower-case names when the fields were created in the database, **defines** must be used to reference them. For example, if a Sybase field is "Last_name" then consider the following **TPL codebook** statements:

1)
```
Last_name CONTROL GET CONDITIONS FROM DATA;
```

2)
```
LAST_NAME DEFINES Last_name CONTROL
        GET CONDITIONS FROM DATA;
```

3)
```
LAST_NAME DEFINES "Last_name" CONTROL
        GET CONDITIONS FROM DATA;
```

Statement (1) is not acceptable since TPL will convert Last_name to LAST_NAME which will not match the database field name.

Statement (2) is also unacceptable since again TPL will convert Last_ name to LAST_NAME

Statement (3) is correct because TPL will not change the case of the quoted item.

ODBC Some databases accessed via ODBC in the Windows version of TPL-SQL are case sensitive in the same way as described for Sybase above. If you get error messages saying that your database fields cannot be found, it may be because you need to enclose the database field names in quotes. If you use the Codebook Builder to prepare your codebook, you do not need to be concerned with this. Codebook Builder will provide the names from the database and enclose them in quotes.

Fields within a single SQL table must have unique names. However, it is common practice for fields in different SQL tables within the same database to have the same name. This is especially true for fields used for TPL associations or SQL joins. When processing a request, TPL Tables and TPL Report must know which SQL table should be used to retrieve data for a variable. There are two options available in TPL Tables and TPL Report. First, you can use the SQL technique of qualifying a name when there is an ambiguity. For example suppose both the **Company** SQL table and the **Employee** SQL table have a field called **company_id**. A TPL Tables request can include a **Table** statement such as:

```
Table T1: Company.company_id by region, total;
```

An alternate approach using TPL would be to use **defines** clauses in your codebook to give the two fields different TPL names.

Creating Subfields with Substr

The **substr** feature lets you describe variables that are subparts of fields in your database. In non-database codebooks, this funtionality is provided by **Redefine**.

The list of data types supported for **Substr** should be the same as the list supported for Control variables.

The syntax for a subfield is:

```
SUBSTR(sql-name, start-position, length-of-subfield)
```

The *sql-name* is the name of the field in the database. If the *sql-name* is not a valid TPL name or is in a database that is case sensitive, it must be enclosed in quotes. In addition, quotes are required for lower case **sql-names**. **Start-position** is relative to the **sql-name** field

An example from a codebook source with subfields is:

```
COMPANY_CODE "Company Code" Defines "company_code" Con from Data
company_type Defines Substr("company_code", 1, 2) Con from Data
Numeric_part Defines Substr("company_code", 3, 3) Con from Data
```

An example of a resolved codebook source with all database-derived information filled in is:

```
COMPANY_CODE "Company Code" Defines "company_code" Con
Right Blank Fill from Data evaluated to Con 5 (
"AM703" = "AM703"
"AP001" = "AP001"
etc.
)
 company_type Defines Substr("company_code", 1, 2) Con from Data
evaluated to Con 5 (
"AM" = "AM"
"AP" = "AP"
etc.
 )
Numeric_part Defines Substr("company_code", 3, 3) Con Right Blank Fill from
Data evaluated to Con 5 (
 "001"  =  "001"
.
.
"703" = "703"
etc.
)
```

Multiple SQL Tables and Association Statements

A SQL database typically has many SQL tables in it. A TPL codebook for a SQL database need not describe all of these SQL tables. However a TPL codebook will usually describe more than one SQL table. The method for describing multiple SQL tables is basically the same as that used for describing multiple record types for a sequential hierarchical file. Separate descriptions are included for each SQL table. In addition, in the SQL case, one or more association statements must be provided to relate the separate SQL tables.

An Example

Consider the Families codebook we discussed earlier. Suppose our SQL database also contains a SQL table of family member data. Our combined codebook source might look like the following:

```
Begin Families Codebook SQL
Family  defines "family" Table
Family_id defines "family_id" obs
Region defines "region" control  get conditions from data
Living_Qrt  defines "living_qrt" "Living Quarters" control from data
Persons_in_family defines "persons_in_famly" obs
Gross_income_of_head  defines "gross_income_of_head" obs
Gross_income_of_spouse defines "gross_income_of_spouse" obs

Member  defines "member" Table
Family_id  defines "family_id" obs
Age defines "age" obs
Sex  defines "sex" control from data
Education defines "education" control from data
Favorite_car defines "favorite_car" from car(car_id,car_name)

Family is parent of member where Family.Family_id = Member.Family_id;
End Families Codebook
```

The example consists of two descriptions of individual SQL tables plus one association statement near the end which relates the two SQL tables. Notice that each of the SQL table descriptions has a field, **Family_id**, on it. This field is used by the association statement to connect the members with their family. The association statement asserts that a **Member** record belongs to a particular **Family** whenever the **family_id** on the **Member** record matches the **Family_id** on the **Family** record. The **parent** tells the system that **Family** is above **Member** in a hierarchical relation. In other words, each **Family** may have multiple **Member**s. TPL imposes few restrictions on the data types of the terms on each side of the equal sign in a **where** clause. However a database error will result if one of the terms is an integer or floating point number in the database and the other contains a non-numeric value.

When TPL Tables or TPL Report processes a request using this association statement, it will read each **Family** record, get the **Family_id** from the record and then use this **Family_id** to retrieve from the **Member** SQL table each of the **Member** records which have this **Family_id**. It will then read the next **Family** record, get its **Family_id** and proceed in the same way.

Warning **Fields on the right side of an association statement should be indexed. Otherwise performance may be unaccepable.** See the section on "Optimizing Performance" for more information.

By default, if a **Family** record has a **Family_id** which appears on no **Member** records, the **Family** record is rejected with an error message. The TPL codebook statements **Process incomplete hiearchies = yes** or **no** and **Report incomplete hierarchies = yes** or **no** will alter this behavior in the same way as it does with sequential hierarchies.

Now suppose we wish to add the **Car** SQL table to our codebook. Each **Member** has one **Favorite_car**. Then we do not have a **parent** relation but rather what we call a **sibling** or **sib** relation. The addition to our codebook might be:

```
Car defines "car" table
Car_id defines "car_id" control from data
Car_name defines "car_name" character
Car_cost defines "car_cost" obs
Car_weight defines "car_weight" obs

Member is sib of Car where Member.Favorite_car = Car.Car_id
```

When TPL Tables or TPL Report processes a database using this new association statement, for each **Member** record it reads, it will determine the **favorite_car** for that **Member**. This **Favorite_car** code will be searched for in the **Car_id** field of the **Car** SQL table. If exactly 1 match is found, processing will proceed normally. If no record in the **Car** SQL table has the appropriate **Car_id** an error message will result and both the **Member** and **Car** records will be discarded. If multiple **Car** records are found to have matching **Car_id**s, an error message will result and the "duplicate" **Car** records will be rejected. Which **Car** record is kept is undefined. If **Report incomplete hierarchies = no** is specified, no error message will be reported for either the case of no **Car** records or multiple **Car** records.

The effect of **Tabulate incomplete hierarchies = yes** depends upon whether a table or report is being created. In a TPL Tables job, **Tabulate incomplete hierarchies = yes** causes a higher level record with no lower level records to contribute to cross tabulations which only involve the higher level records. But when we are using **sib** associations, both records are at the same level. So there can be no cross tabulations to which the **Member** record contributes and the **Car** record does not. Thus **Tabulate incomplete hierarchies = yes** has no effect on records in an incorrect **sib** association in TPL Tables job.

In a TPL Report job, **Tabulate incomplete hierarchies = yes** results in a report row being generated for a record which does not have a sibling. The sibling columns for that row are marked as missing data.

It is not an error if multiple **Members** have the same favorite car or if some car is the favorite of no members. All that is required for a correct **sib** association is that each record from the SQL table to the left of the **sib** association have exactly one match in the SQL table to the right of the **sib** association. This brings up an important point that will be discussed more fully later. When a table or report request is processed, the resulting output will depend upon which associations are used. In our example, if we compute the average cost of **favorite_car**s by using our **sib** association to get from the **Member** SQL table to the **Car** SQL table we will not

get the same result as we would if we just calculated the average cost of cars in the **Car** SQL table. In the former case, the average is weighted by how many members have a car as their favorite. In the latter case, the average is unweighted.

More on Association Statements

In the above example, **Member** and **Car** were associated using a single pair of fields. In some cases multiple pairs of fields must be used to specify an association. These pairs are connected by **and**. Consider a database of **Employer**s and **Employee**s. Assume some of the **Employer**s have many branches. An **Employee** is employed at a single branch. The branch is designated by both a **Company_id** and a **Branch_id**. Then we might have two different association statements to relate the **Employee** SQL table to the **Employer** SQL table:

```
Employer is parent of Employee where
Employer.Company_id = Employee.Company_id

Employer is parent of Employee where
Employer.Company_id = Employee.Company_id
and Employer.Branch_id = Employee.Branch_id
```

A table or report request may use either of these associations. If the first one is used and we calculate the average number of employees per employer we will get the average number of employees per company. If we use the latter association statement we will get the average number of employees per branch.

As a notational convenience, the association statements can be written more concisely as:

```
Employer is parent of Employee where Company_id = Company_id

Employer is parent of Employee where Company_id = Company_id
and Branch_id = Branch_id
```

The SQL table name to the left of an equal sign in a **where** clause is assumed to be the same as the SQL table name to the left of **parent** or **sib** and the SQL table name to the right of the equal signs is assumed to be the same as the SQL table name to the right of **parent** or **sib**.

Use of %INCLUDE in Codebooks

A TPL codebook describing a large complex database can become very long. An easy way to make the codebook more manageable is to create separate files for descriptions of each SQL table. You can then use the **%INCLUDE** feature to as-

semble these files into a single TPL codebook. This technique is especially useful for corporate databases which are used by many different categories of users. Certain users will produce tables or reports from only some of the SQL tables while other users will use a different set of SQL tables for their work. Separate TPL codebooks may be made for these different users by including only those SQL table descriptions that they will need for their work.

Codebook Abstract

When the TPL codebook processor is run against a SQL database, multiple files are produced. For MS Windows, one is the new codebook source with **evaluation**s in it. This was discussed earlier. The second file is the **.K** file which is the codebook executable. This is a file that is not intended for reading. The third file is the **.L** file which contains error messages for an unsuccessful codebook or the **Abstract** of a successful codebook run. The Abstract for a TPL-SQL codebook is similar to the Abstract produced from a sequential file with a few exceptions.

A codebook for a sequential file contains columns for the number of bytes described, the level, and the parent of each record. None of these are relevant to a SQL database record. The sequential file codebook also contains information on the location of each field within a record. This is also not relevant for a SQL description.

The TPL-SQL codebook abstract contains two things not found in a sequential file codebook. One is the SQL column name. This name will match the TPL variable name unless a **defines** has been used to change the TPL name.

The second addition to a TPL-SQL codebook abstract is a list of the Association statements that have been specified. Each of these association statements has a number assigned to it. This number is important. It may be needed in creating a plan for processing the data in a table or report request.

```
TPL CODEBOOK Copyright(C) 2005 QQQ Software, Inc. All Rights Reserved.Version 6.0 of CO-
DEBOOK compiled on Wed May 11 18:03:09 EST 2005.

TPLDB ABSTRACT FOR DATABASE tpldb
Created 6/17/05 at 4:49:16 PM from codebook source tpldb.cbk

The records and variables described in your codebook are listed below
in alphabetical order.  For non-database codebooks, the first position
of each record is location 1.

RECORD SQL TABLE

OFFICE          BRANCH
COMPANY         COMPANY
```

EMPLOYEE PERSON

SQL DATABASE ASSOCIATIONS

1: OFFICE is parent of EMPLOYEE where ID = ID
2: COMPANY is parent of OFFICE where COMPANY_ID = COMPANY_ID and BRANCH = BRANCH

VARIABLE	SQL COLUMN	SIZE	TYPE	TPL RECORD
NUMBER_EMP	SIZE	4	OBS	OFFICE
OFFICE	BRANCH	—	RECORD OBS	OFFICE
BIRTH	BIRTH	—	CON	EMPLOYEE
BIRTH_C	BIRTH	—	CHAR	EMPLOYEE
BIRTH_O	BIRTH	—	OBS	EMPLOYEE
BRANCH	BRANCH	4	OBS	COMPANY
BRANCH	BRANCH	4	OBS	OFFICE
COMPANY	COMPANY	—	RECORD OBS	COMPANY
COMPANY_CHAR	COMPANY_NAME	20	CHAR	COMPANY
COMPANY_ID	COMPANY_ID	4	OBS	COMPANY
COMPANY_ID	COMPANY_ID	4	OBS	OFFICE
COMPANY_NAME	COMPANY_NAME	20	CON	COMPANY
ID	IDX	1	OBS	OFFICE
ID	ID	10	OBS	EMPLOYEE
PERSON	FULLNAME	20	CON	EMPLOYEE
SALARY	SALARY	8	OBS	EMPLOYEE
SEX	SEX	1	CON	EMPLOYEE
EMPLOYEE	PERSON	—	RECORD OBS	EMPLOYEE

End CODEBOOK processing

TABLE AND REPORT REQUESTS FOR SQL DATABASES

A TPL Tables or TPL Report request run against a SQL database looks very much like a request run against a sequential file. There are five main differences:

1) The command line for invoking the job is slightly different. (The command line options appropriate for your database system are described in the run instructions appendices.)

2) A job run against a SQL database may require that some variable names be qualified with the name of their SQL table.

3) A SQL database request may include Association statements.

4) A request run against a SQL database may require a plan to specify how the data is to be read.

5) Statements can be included to optimize performance. A table or report request run against a SQL database may include a **SQL Select** statement in addition to or instead of a regular TPL **Select** statement. You can use a **SQL Fetch** statement in your profile or format request for additional performance tuning.

Qualified Names

All TPL variable names for sequential file fields must be unique. As was stated earlier, for a SQL database, the TPL names need be unique only for a given SQL table. TPL must know which SQL table it should retrieve data from. If there is an ambiguity, the variable names must be qualified by the TPL name for the SQL table; e.g. **Employer.Branch_id** or **Employee.Branch_id**. TPL Tables and TPL Report will produce an error message whenever an ambiguity exists.

The need for qualified names can be completely eliminated by using **defines** clauses in the codebook to assign unique TPL names to each database field. If you have not done this, TPL still minimizes the need for qualified names by the following procedure.

Suppose you wish to use **Branch_id** which is found on both the **Employee** SQL table and the **Employer** SQL table. The first time you use **Branch_id** in your request you must qualify it; e.g. **Employee.Branch_id**. From that point forward, TPL will assume that when you use **Branch_id** you mean **Employee.Branch_id**. If in fact you wish to use **Employer.Branch_id**, you must explicitly qualify the name. From that point forward, you must qualify all occurrences of **Branch_id**. If this rule seems complicated, don't worry. You may choose to always qualify ambiguous TPL names. Also, if you fail to qualify a name that requires qualification, TPL will produce an error message rather than risk making an incorrect assumption about what you intend.

Association Statements in Table or Report Requests

In some cases, you may wish to create a table or report which requires an association of SQL tables which was not anticipated when the TPL codebook was created. Rather than require that the codebook be recreated, TPL allows you to add association statements to a TPL Tables or TPL Report request. Association statements in table and report requests are the same as association statements in the codebook except that a semicolon (**;**) is required at the end of each association statement. Association statements, if included in a request, must be located immediately after

the **Use** statement of the table or report request. They are assigned numbers in order of occurrence beginning after the last number used by the codebook association statements. For example, if the codebook has 3 association statements and a table request has 2 more, the two table request association statements will be numbered 4 and 5.

The Processing Plan

By far the most important difference between a TPL Tables or TPL Report request run against a SQL database and against a sequential file concerns the sequence of records delivered to TPL for processing. In processing a sequential file the data records are delivered to TPL in precisely one possible order — the order the records appear on the file. In the database case, the records may be delivered to TPL in a variety of orders. The "same" request can produce radically different output if a different sequence of records is delivered to TPL.

The sequence of records delivered to TPL is determined by a **Plan**. A Plan is a list of association statements which "chain" together the SQL tables. A plan is **valid for a request** if it satisfies a collection of conditions:

1) All variables used in the request must be on SQL tables chained together in the plan.

2) No SQL table may be visited more than once in following the plan chain.

3) The plan must define a single hierarchical path through the database.

The next few sections will discuss these conditions.

What is a Chain?

SQL tables are chained together by a set of associations if starting from some node SQL table, we can get from it to each of the other SQL tables in the chain by going from the SQL table on the left of an association to a SQL table on the right of an association. For example suppose we have the following associations (the where clauses have been omitted for clarity):

```
1:      A is parent of B where ...
2:      B is parent of C where ...
3:      B is sib of G where ...
4:      G is parent of F where...
5:      C is parent of D where ...
6:      E is parent of D where...
7:      D is parent of F where ...
```

Starting with **A** as our node we can get to **B** (using 1) then **C** (using 2) then **D** (using 5) then **F** (using 7). So **A→B→C→D→F** form a chain. We can also form the chains **E→D→F** and **A→B→G→F** and several others. There is no chain which includes both **A** and **E**. So a table request which uses data from SQL table **A** and SQL table **E** could not be processed using the collection of Association statements listed.

How Can A SQL Table Be Chained to Itself?

The requirement that a plan not pass through the same SQL table twice is because TPL would not know which pass should be used to evaluate a variable. In some database formats people do wish to violate this rule. Suppose for example you have all employees including supervisors in the same file. You wish to have a count of how many employees have supervisors who earn particular salaries. You would like to use the Association:

```
Employee is parent of Employee where employee_id = is_supervised_by;
```

The solution to this problem is to use the **defines** construct in the codebook. In our previous examples we have assigned a new TPL name to a SQL field. We can also use a **defines** to assign a new name for a SQL table. Our codebook would include the following statements:

```
Employee Table
    description of employee fields
Supervisor defines Employee Table
    same description of fields
Supervisor is parent of Employee where employee_id = is_supervised_by
```

What is a "Single Hierarchical Path"?

Probably the easiest way to identify a "single hierarchical path" is to assign level numbers to the SQL tables in a plan. The node of the plan is the SQL table on the left of the first association statement. It is assigned level 0. Follow the plan chain. If the association statement contains **sib** the SQL table on the right is given the same level number as the SQL table on the left. If the association statement contains **parent** then the SQL table on the right is given a level number 1 higher than the SQL table on the left. Now look at your completed list. If any two **parent** associations have SQL tables with the same level number on the left, the plan does not define a single hierarchical path. Consider the association statements we looked at earlier:

```
A is parent of B where ...
B is parent of C where ...
B is sib of G where ...
```

G is parent of F where...
C is parent of D where ...
E is parent of D where...
D is parent of F where ...

One plan we specified was **A→B→G→F**. If we apply our test to this plan we get:

A **[level 0]** is parent of B **[level 1]**
B **[level 1]** is sib of G **[level 1]**
G **[level 1]** is parent of F **[level 2]**

No two parent associations have the same level number so the plan is a single hierarchical path.

Another valid plan is: **A→B→G→C** which uses the rules

A **[level 0]** is parent of B **[level 1]**
B **[level 1]** is sib of G **[level 1]**
B **[level 1]** is parent of C **[level 2]**

Compare this with the chain **A→B→C→G→F**. This chain uses

A **[level 0]** is parent of B **[level 1]**
B **[level 1]** is parent of C**[level 2]**
B **[level 1]** is sib of G **[level 1]**
G **[level 1]** is parent of F **[level 2]**

In this example, the second and fourth associations both are parent relations starting at level 1 so we do not have a single hierarchical path.

Why Does TPL Need a Single Hierarchical Path?

An example is probably the easiest way to answer this question. Suppose we have a database with 3 SQL tables. The SQL tables are **Company**, **Company_car**, and **Employee**. These SQL tables are connect by:

```
Company is parent of Company_car where company_id = owner_id
Company is parent of Employee where company_id = employer_id
```

These association statements jointly define a plan which violates the single hierarchy requirement. Now consider the following table request:

```
Compute Ratio = Car_price / Employee_salary;
Post compute Ave_car_salary_ratio =Ratio / Company_car;
Table T1: Company_location, Ave_car_salary_ratio;
```

This table request should tell us something about the cost of keeping employees happy in different locations. Unfortunately, TPL Tables cannot process this request. The problem is that there is no way to compute **Ratio** since we cannot pair an **Employee_salary** with a particular **Car_price**. We don't know who drives which car.

Let's change the example slightly. Suppose each company buys only one model of car. Then our association statements become:

```
Company is sib of Company_car where company_id = owner_id
Company is parent of Employee where company_id = employer_id
```

Now our plan is valid and our table request works. We know which car the employee has because we know his company and which car the company buys. So we can compute **Ratio**.

TPL doesn't allow multiple hierarchical paths in a single table or report request because the TPL system then does not have enough information to combine fields from the different paths.

Plan Selection

When the TPL Tables or TPL Report request processor encounters a request which is to be run against a database, the system analyzes the request and the association statements in the request and codebook. If you have specified a plan, it tests whether the plan you specified is valid for your request. If you have not specified a plan, it determines all valid plans for the request. If there is exactly one valid plan, the plan is reported at the end of the translation step and processing continues. If there is no valid plan or there are multiple valid plans, processing stops.

When there is no valid plan for your request, you have two choices. First you may add additional association statements to your request or codebook so that all required SQL tables are chained into a single plan. If this cannot be done, you must modify your request by eliminating all references to fields on the SQL tables which cannot be linked into the plan chain. Sometimes splitting a job into two separate jobs will enable you to get all of the data you want from the database while using valid plans.

If your request can be processed using more than one valid plan, the TPL system will list all valid plans and stop at the end of the translation step. At this point you must examine the listed plans and determine which if any correctly capture the desired meaning of your tables or reports. You must then add a plan statement to your request and reprocess the request. The following is an example of the output at the end of the translation step:

```
PLANS:

Read:    COMPANY
  3:     COMPANY is parent of OFFICE      where COMPANY_ID = COMPANY_ID and
         BRANCH = BRANCH
  2:     OFFICE      is sibling of OFFICE_1    where COMPANY_ID = COMPANY_ID and
         BRANCH = BRANCH
  1:     OFFICE      is parent of EMPLOYEE     where ID = ID

Read:    OFFICE
  2:     OFFICE      is sibling of OFFICE_1    where COMPANY_ID = COMPANY_ID and
         BRANCH = BRANCH
  1:     OFFICE      is parent of EMPLOYEE     where ID = ID
  4:     EMPLOYEE    is parent of COMPANY      where ID = OWNER

Read:    EMPLOYEE
  4:     EMPLOYEE    is parent of COMPANY      where ID = OWNER
  3:     COMPANY is parent of OFFICE      where COMPANY_ID = COMPANY_ID and
         BRANCH = BRANCH
  2:     OFFICE      is sibling of OFFICE_1    where COMPANY_ID = COMPANY_ID and
         BRANCH = BRANCH

*** ERROR:  Since there is more than one possible plan for processing
this request, you must select one of the above plans by inserting a
plan statement in your request.  Suppose the plan you wish to use has
the numbers 3,1,2 in that order next to the association statements.
Then your plan statement would be:

                PLAN 3 1 2;
```

How to Specify a Plan

A plan specification is just the word **PLAN** followed by a list of association statement numbers (in processing order) followed by a semicolon (**;**). An example of a plan statement is:

```
PLAN  3 5 12;
```

Association statement numbers may be obtained from the list of valid plans as in the example above. Alternately, they may be obtained from the codebook abstract. As mentioned before, if you add new association statements to your table or report request, their statement numbers are just the next unused numbers. A **PLAN** statement may occur anywhere in a table or report request after the **USE** statement and any association statements.

Plans and the COUNT Variable

Count is a built-in variable in TPL Tables. If a cross-tabulation has no explicit observation variable, **Count** is implicitly taken to be the observation variable. In a sequential hierarchical file **Count** gives a count of the number of records at the lowest level of the hierarchy. In a table request run against a SQL database, **Count** gives the equivalent result — a count of the number of records accessed from the SQL table on the right of the last association statement in the plan. The danger inherent in this is that if the plan changes, the count will also change.

Suppose we have a database with **Industry**, **Company**, and **Employee** SQL tables. The SQL tables use the associations:

```
1:    Industry is parent of Company where industry_id = industry_id
2:    Company is parent of Employee where company_id = company_id
```

We produce a table statement:

```
Table T1: Industry_category, Company_location;
```

Assuming **Industry_category** is on the **Industry** SQL table and **Company_location** is on the **Company** SQL table, the plan is just:

```
Plan 1;
```

Thus the table will be a count of companies for each location and industry category. Now suppose we add a second table to our request:

```
Table T2: Industry_category, Education_level;
```

where **Education_level** is on the **Employee** SQL table. Our request now requires the plan:

```
Plan 1 2;
```

The implicit **Count** now counts employees. Without changing the first table, we have changed its meaning. We now get a count of employees for each industry category and location instead of a count of companies.

The safest way to avoid problems in counting is to **always explicitly include the SQL table name in any cross-tabulations that do not already have an observation variable**. Then you will know exactly what you are counting.

Optimizing Performance

Indexing for Multi-Table Processing

Fields on the right side of an association statement should be indexed in the database. This is true for sibling, one-to-one associations as well as for hierarchical, one-to-many associations.

TPL does not use joins to process multiple SQL tables. Instead it processes the data in a hierarchical fashion. Suppose you have a database with **Employer** and **Employee** SQL tables. The tables are in a parent-child relationship where matches are on the basis of **employer_id** on each of the SQL tables. You wish to produce a TPL table using both of these SQL tables. TPL will read the first **Employer** record and find the value from that record for the **employer_id** field. It will then search through the **Employee** SQL table for each **Employee** record with the desired **Employer_id**. If your database does not have an index built on **Employer_id** on the **Employee** SQL table, then TPL will have to read through the entire **Employee** SQL table for each **Employer**. This can produce unacceptable performance!

In order to avoid this performance problem, your database must have indexes built on the key fields used on the "child" or right-hand side of the Association statement. If multiple key fields are needed to relate two SQL tables, you will get the best performanace if your database has an index based on the combined fields.

SQL Select

A table or report request run against a SQL database may include a **SQL Select** statement in addition to or instead of a regular TPL **Select** statement. The **SQL Select** statement provides support for an optimization which sometimes produces significantly improved performance. If you use a regular TPL **Select** statement in your table or report request with a SQL database, all records which follow the plan are delivered to TPL for processing. Those which fail the **Select** are rejected by TPL. If you use a **SQL Select** statement, records are rejected within the database software. Use of this statement improves performance by reducing network traffic and by saving TPL from processing data which it does not need.

Importance of Indexing and an Efficient SQL Select Statement

If the **SQL Select** statement excludes a large share of the data, a significant time savings can result. However, you must be careful that your **SQL Select** statement is efficient. TPL passes the **SQL Select** statement to the database system "as is" without attempting to optimize it.

SQL Select improves performance only if doing selection within the database is as fast as doing it within TPL. If you are selecting on a non-indexed field, the database selection is usually much slower than TPL selection. So don't use **SQL Select** if the field being selected on is not indexed.

In one case a user had 300,000 establishment records in his database. He had an indexed field **cycle** on **Establishment**. He first tried the **SQL Select** statement:

```
SQL Select on establishment "cycle between 115 and 123";
```

Using this statement, his request took 2 hours.

He replaced his **SQL Select** statement with:

```
SQL Select on establishment
 "cycle in (115,116,117,118,119,120,121,122,123)";
```

This TPL request produced the same results but took only 5 minutes to process.

The reason why the second one was so much faster is that since **cycle** was indexed, the individual values in the **in** clause could be found quickly while the **between** construction required the database to do a sequential search for values in the range.

Description of SQL Select

The syntax of a **SQL Select** statement is:

```
SQL Select on SQL-Table "selection-string";
```

SQL-Table is the TPL name for a SQL table. *selection-string* is a string of text which is appended unchanged to the **where** clause of a **SQL Select** statement. Since the *selection-string* is not modified by TPL, it should contain SQL field names rather than TPL variable names.

Suppose the **Age** field is on the SQL table **Person**. Our TPL codebook has used a **defines** clause to assign the TPL name **Age_obs** to **Age**. Then the following two statements should produce the same result:

```
Select if Age_obs < 50;
SQL Select on Person "Age < 50";
```

Sybase and ODBC String values passed in SQL select statements must be in single quotes; e.g.

```
SQL Select  on Company  "name = 'QQQ Software' ";
```

If TPL and the database software are both running on the same computer, there will be little difference in performance between having TPL reject records and having the database software reject the records. Bigger differences will occur if TPL and the database software are running on different machines and if the selection is to be done at the bottom level of the hierarchy defined by the plan. In such cases data will be rejected before it travels across your network. If the machine running the database software is faster than the machine running TPL, additional performance improvements will be realized.

In most table and report requests, far more records are retrieved at the bottom level of the processing hierarchy than at higher levels. If a record fails a select at a level above the bottom level of the hierarchy, then no records will be retrieved from the database from lower levels regardless of whether a TPL **Select** or a **SQL Select** is used. Thus if selection is done above the bottom level of a hierarchy, there is unlikely to be much difference in performance between using a regular TPL **Select** and a **SQL Select**.

Difference in Results between Regular Select and SQL Select

In rare cases, **SQL Select** and regular **Select** can produce different results. The differences only occur when **Tabulate Incomplete Hierarchies = Yes;** has been specified.

Suppose we have a database with **Family** and **Member** data. If a family has no members, it will still contribute to the table if **Tabulate Incomplete Hierarchies = Yes;** is specified. Instead, suppose the database family does have members but a **SQL Select** is used to remove all of its members. To TPL, the cases are the same and the family will contribute to the table if **Tabulate Incomplete Hierarchies = Yes;** is specified.

Now suppose the database family does have members but a regular **Select** is used to remove all of them. TPL requires that an entire hierarchical unit pass a **Select** in order for any part of it to be included in the tabulation (see the section on the Select statement in the Hierarchies chapter). So the family is excluded regardless of the setting for **Tabulate Incomplete Hierarches**. In this rare case, **SQL Select** allows a family to be included which a regular **Select** excludes.

SQL Fetch

The **SQL Fetch** statement is a tuning parameter which should be placed in your **profile.tpl** file or in your format statements. It affects the amount of data that is moved from the SQL Server to TPL on each request for data.

The syntax of a **SQL Fetch** statement is:

```
SQL Fetch count = n;
```

where *n* is an integer. The default is 10.

In cases where TPL is executing on one machine and your database is on another, the choice of value can strongly affect network traffic and performance. In a typical example, changing the **SQL Fetch Count** value from 1 to 10 caused the job to run in 1/3 of the time!

Choosing too high a value for **SQL Fetch Count** will cause the job to use more memory than needed. This can actually slow down performance. Too small a value will degrade performance. If your table or report request uses a single SQL relation, then there is no theoretical limit to how high a value you can use. However there is probably little to be gained by using a value greater than 100. If you are processing a database hierarchically, select a **SQL Fetch Count** value no larger than the largest number of records at the bottom of the hierarchy associated with any given record immediately above. For example, if you are processing a family-member hierarchy and no family has more than 12 members, then 12 is the ideal choice for **SQL Fetch Count**. The exact choice is not critical. There will be little performance difference if you use 15 or 8.

SUMMARY

TPL-SQL provides TPL Tables and TPL Report with direct access to data stored on a SQL database. No intermediate storage is required. You do not need to know SQL in order to user the interface.

A TPL-SQL codebook is a simplified standard TPL codebook. It differs from a standard codebook in that information such as field widths can be omitted because these can be obtained from the database itself. The one important addition found in TPL-SQL codebooks is association statements which specify how different SQL tables are to be processed together. These association statements are chained together to form a plan for reading through the data during processing of a table or report request.

A TPL-SQL table or report request is also very similar to a standard TPL Tables or TPL Report request. The primary difference is that you may need to select the plan that is to be used in processing the data.

CHAPTER 21

Format

THE FORMAT LANGUAGE

Introduction

The FORMAT language gives you precise control over the format of your reports. The automatic formats provided by TPL REPORT are usually acceptable for a quick look at your data. However, presentation or publication standards in your organization may require that you adjust your report formats in ways that cannot be achieved by using TPL statements alone. In other cases, you may find that the default values for such things as column widths or page size are not appropriate for the types of reports you are doing. These defaults can be changed with FORMAT statements.

Several FORMAT statements apply only to the preparation of reports in PostScript mode. With PostScript, you can choose proportional fonts, type sizes and colors for different parts of your reports. PostScript reports can be printed on any laser printer or typesetting machine that processes PostScript code. A separate chapter contains instructions on getting started with PostScript. If you do not have a way of printing PostScript reports, you can ignore the statements that apply to PostScript only.

A special FORMAT statement called **DATA REPORTS** can be used to format your reports as a data file that can be used as input to other types of software, such as TPL TABLES, spread sheets or graphics programs.

Note for Users of TPL TABLES

TPL REPORT and TPL TABLES share the same FORMAT language. Some of the statements do not behave in exactly the same way in both, because of the differences between reports and tables. In addition, some of the statements only apply to one of the two: reports or tables.

All statements in the TPL profile are FORMAT statements. Since the FORMAT language is shared, you can use the same TPL profile for both TPL TABLES and TPL REPORT.

If a statement that applies only to tables is included in a report job, it is ignored by TPL REPORT. For example, a statement such as REPLACE WAFER LABEL has no meaning in TPL REPORT, because reports don't have wafers.

The word TABLE is replaced by the word REPORT when it is read by TPL REPORT. For example, if you use the DATA TABLES statement with TPL REPORT, you will get DATA REPORTS.

Where to Put FORMAT Statements

FORMAT statements are prepared using an editor and saved in a file called a format request. The format request can be used along with the TPL report request file.

FORMAT statements can also be included in your TPL profile. This is a good approach if you want certain statements to apply to all of your reports whenever you run a TPL REPORT job.

Composition of FORMAT Statements

A FORMAT statement consists of two parts: a FOR clause and an ACTION clause. The ACTION clause specifies what is to be done to the reports. The FOR clause specifies where the ACTION clause should take effect.

A typical FOR clause is:

```
FOR REPORTS 1 TO 3  COLUMNS 1, 3 AND 5 :
```

Some typical ACTION clauses are:

```
COLUMN WIDTH = 14;
TOP MARGIN = .5 IN;
REPLACE MASK WITH 99,999.99;
PAGE WIDTH = 140;
```

FOR clauses are optional. If there is no FOR clause before an ACTION, the FOR clause from a previous statement applies to the new ACTION. If there is no previous FOR clause, the ACTION applies to the entire set of reports in the request.

The following set of FORMAT statements shows how ACTIONS can be grouped with FOR clauses. The first two statements apply to all reports unless specifically changed by subsequent FOR clauses.

```
PAGE WIDTH = 120;
FOR REPORT 1 :
        REPLACE TITLE WITH 'Sales results sorted by region.';
        COLUMN WIDTH = 20;

FOR REPORT 2 COLUMNS 1, 3, AND 5:
        COLUMN WIDTH = 12;
        REPLACE MASK WITH 9,999,999;

FOR REPORT 3 VARIABLE DOLLARS:   REPLACE LABEL WITH 'Revenue';
```

Action Levels

Different types of ACTION clauses take effect at different levels: **request**, **report**, **column**, or **cell**.

For example,

```
PAPER = LETTER;
```

is applicable at the report level; that is, you cannot specify different paper sizes for different reports created by the same report request. On the other hand, COLUMN WIDTH can be specified for individual columns within individual reports.

If any part of a FOR clause is inapplicable for an associated ACTION because of the level of the ACTION, the term in the FOR clause is ignored. Consider the FORMAT statement:

```
FOR REPORT 3  COLUMNS 1 TO 6 :
        RETAIN CROSS RULES;
```

This statement will cause horizontal rules (lines) to be inserted above and below the column labels and at the bottom of report 3. The column restrictions are inapplicable, so they will be ignored. If REPORT 3 were omitted from the FOR clause, cross rules would be inserted in all reports in the request.

When an ACTION is specified with a FOR clause that does not apply, TPL REPORT follows the statement with a message in the OUTPUT file. If you find that some of your FORMAT statements are applied (or not applied) in the way that you expect, check the OUTPUT file for messages.

Action Conflicts

If two or more conflicting actions are specified for the same part of a request, the last one specified will win. An example of actions in conflict is two column widths specified for the same column:

```
COLUMN WIDTH = 12;
FOR REPORT 1  COLUMNS 1 TO 5:  COLUMN WIDTH = 8;
```

In this case, all columns in the request will have a width of 12 except columns 1 to 5 in report 1. Those columns will have a width of 8.

Action Size Specifications

For any action that specifies a size, the size is specified by

```
amount  [ unit ]
```

where amount is a number and unit is optional. If no unit is specified, characters are assumed. If a unit is specified, the amount can be a decimal number and the unit can be expressed as inches, centimeters or points using any of the following words or abbreviations:

```
inch
inches
in
ins
cm
points
pt
pts
```

Fractional sizes must be specified as decimal numbers. For example,

```
STUB WIDTH = 2.5 IN;
```

What can be in the FOR Clause?

The following elements can be referenced in a FOR clause:

```
REPORT
COLUMN
VARIABLE
CONDITION
```

REPORT and COLUMN can be referenced by number. REPORT, VARIABLE and CONDITION can be referenced by name.

Note *Report rows cannot be referenced in FOR clauses.* Row references, such as FOR ROWS 3 TO 6, are ignored. Some statements, such as SKIP LINE EVERY and RULE EVERY can perform actions on rows without specific row references.

Variables can be specified by

```
FOR VARIABLE variable name:
```

Variable references only have meaning **for actions applied to columns**.

Control variable conditions can be specified by

```
FOR CONDITION variable name(condition number):
        or
FOR CONDITION variable name(condition name):
```

Condition references only have meaning **for actions applied to values in report cells**.

Multiple variables or conditions can be referenced in the same FOR clause, with or without commas between them. Examples are:

```
FOR VARIABLE A VARIABLE B :
FOR VARIABLES A, B, C :
FOR CONDITIONS VAR(1), VAR(2), VAR1(1) :
FOR CONDITIONS VAR1(1,2), VAR2(1) :
```

Reports can be specified by

```
FOR REPORT report name(s):
        or
FOR REPORT report number(s):
        or
FOR REPORTS ALL:
```

Ranges of values can be expressed in the FOR clause using the word TO. Commas, equal signs, and the word AND are optional. For example,

```
FOR REPORT C3, COLUMN = 1  AND COLUMNS 4 TO 6 :
```

means the same as

```
FOR REPORT C3   COLUMNS   1  4  5  6 :
```

FOR clauses can include increments. For example,

```
FOR COLUMNS 1 TO 8 BY 2 : .....
```

This clause means: In the range 1 to 8, begin with column 1 and take every 2th column. It means the same as the following clause.

```
FOR COLUMNS  1  3  5  7 : .....
```

The Format Actions

The FORMAT ACTIONS are grouped by type and listed below. In the FORMAT reference section of this chapter, statements are ordered alphabetically and described in detail.

Note All FORMAT statements must end with a semicolon (;).

Control Page Size

```
PAGE LENGTH = size;
PAGE LENGTH = AUTOMATIC;
PAGE WIDTH  = size;
PAGE WIDTH  = AUTOMATIC;
PAPER = paper-type;
```

Change Column Widths

```
COLUMN WIDTH = size;
COLUMN WIDTH = AUTOMATIC;
COLUMN WIDTH = AUTOMATIC MAXIMUM = size;
```

Delete or Retain a Report or Part of a Report

```
DELETE      ALL RULES;
or          BLANKS;
RETAIN      COLUMNS;
            CROSS RULES;
            DOWN RULES;
            HEADING;
            LEADING ZEROS;
            REPORTS;
            SIDE RULES;
            TITLE;
```

Rules and Blank Lines

RULE EVERY n;
SKIP LINE EVERY n;

Bank Columns

BANK AFTER COLUMN;
BANKS PER PAGE = n;
SKIP n LINES AFTER BANK;

Margin Sizes *(see MARGIN)*

LEFT MARGIN = size;
RIGHT MARGIN = size;
TOP MARGIN = size;
BOTTOM MARGIN = size;

Mark Pages with Page Numbers and Other Information

PAGE MARKER marker specifications;
BOTTOM PAGE MARKER = marker specifications; (PostScript only)

Align Left, Right or Center

ALIGN CELLS direction;
ALIGN COLUMN direction;
ALIGN HEADING LABELS direction;
ALIGN REPORT direction;
ALIGN TITLE direction;

Control Position of the NUMBER Column

NUMBER RIGHT;
NUMBER LEFT;
NUMBER BOTH;

Replace Labels, Masks and Values

REPLACE LABEL WITH label;
REPLACE MASK WITH mask;
REPLACE TITLE WITH label;
REPLACE TITLE CONTINUATION WITH label;
USE CONDITION LABEL;
USE CONDITION NAME;
USE CONDITION VALUE;
USE VARIABLE NAME;
USE VARIABLE LABEL;

Replace Column Divide Character

```
REPLACE DIVIDE CHARACTER WITH 'char';
```

Format Reports as a Data File

```
DATA REPORTS;
DATA REPORTS ZERO FILL;
```

PostScript

```
POSTSCRIPT = YES;  (or NO)
FONT = type/size;
ROTATE;
EXTRA LEADING = n;
RULE WEIGHT = n;
DOWN RULE WEIGHT = n;
REPLACE MASK FONT WITH font;
RETAIN CROSS RULES  WEIGHT = n and/or DOUBLE;
RETAIN SIDE RULES  WEIGHT = n;
```

Color PostScript

```
COLOR defaults:
        DEFAULT COLOR = color;
        LABEL COLOR = color;
        RULE COLOR = color;
        SYMBOL COLOR = color;
COLOR = NO;  (or YES)
REPLACE COLOR color WITH FONT font;
REPLACE MASK COLOR WITH color;
```

Print and Export Control (*UNIX only*)

Normally (in default mode), the system will prompt you at the end of a job to find out whether you want to print outputs or export files to other formats. You can use the following statements to select the print options in advance.

```
CSV OUTPUT = YES or NO or PROMPT;
EPS OUTPUT = YES or NO or PROMPT;
PRINT OUTPUT = YES or NO or PROMPT;
PRINT TABLES = YES or NO or PROMPT;
```

The default for all statements is PROMPT.

The NUMBER Variable in FORMAT Statements

The NUMBER variable is a special built-in variable that contains the row number for each row of data in a report. By default, a NUMBER column is included as the first column for each page of report output, headed by the label "Row". The NUMBER column does not have a column number, but it can be referenced by name in FOR clauses.

NUMBER can be used in the following FORMAT statements:

```
FOR VARIABLE NUMBER:  DELETE COLUMN;
FOR VARIABLE NUMBER:  RETAIN COLUMN;  (the default)
FOR VARIABLE NUMBER:  REPLACE MASK WITH mask;
FOR VARIABLE NUMBER:  REPLACE LABEL WITH label;  (default is "Row")
FOR VARIABLE NUMBER:  ALIGN COLUMN LEFT, RIGHT or CENTER; (the default)
FOR VARIABLE NUMBER:  ALIGN CELLS LEFT, RIGHT or CENTER; (the default)
FOR VARIABLE NUMBER:  ALIGN HEAD LEFT, RIGHT or CENTER; (the default)
FOR VARIABLE NUMBER:  COLUMN WIDTH = n;
NUMBER RIGHT;
NUMBER LEFT;  (the default)
NUMBER BOTH;
```

NUMBER RIGHT and NUMBER LEFT specify whether the NUMBER column should be on the right or left of the other columns in the report. NUMBER BOTH means that the NUMBER column should be put on both sides.

Examples

```
NUMBER RIGHT;
FOR REPORT 1, VARIABLE NUMBER:
        REPLACE LABEL WITH 'Row Number";
FOR REPORT 2 VARIABLE NUMBER AND COLUMN 3:
        COLUMN WIDTH = 6;
```

Use of FORMAT Statements in Profile

Any FORMAT statement can be included in your TPL profile (the file called **profile.tpl**). These statements will be the default values for all runs, or, if you have a profile in your current directory, the statements from that profile will determine the defaults for all jobs run from that directory.

In addition to the standard FORMAT statements, there are a few statements which are peculiar to the profile. The EDITOR statements are initially set when you install TPL REPORT.

International Formats and Non-English Alphabets *(profile only)*

```
CODEPAGE = name;
COUNTRY = name;
```

PostScript Display *(profile only)*

```
DISPLAY NAME = PostsScript-displayer;
```

Editor Specifications (*UNIX profile only*)

```
EDITOR NAME = editor_name ;
EDITOR FILE = editor_file ;
```

Other statements can be added to the profile to control what information is displayed on the screen and whether the output should be printed.

Printer Selection (*UNIX profile only*)

```
PRINT COMMAND = 'command';
```

FORMAT LANGUAGE REFERENCE

Introduction

In the preceding section, we have provided an overview of the FORMAT language. In this section, we describe each FORMAT statement in detail. The statement descriptions are arranged in alphabetical order.

ALIGN CELLS

Format There are three possible alignments for data cells.

```
ALIGN CELLS LEFT;
ALIGN CELLS RIGHT;
ALIGN CELLS CENTER;
```

Meaning The entries in the report columns, called **cells**, can be aligned to the left, right or center. If a cell entry is more than one line long, all lines of the entry will be aligned the same way.

You will find this statement particularly useful if you wish to change the default alignment for CONTROL and CHAR values and for changing the alignment of OBSERVATION values without entering alignments into individual masks. If both a mask and an ALIGN CELL specification apply to the same column containing values for an observation variable, the last specification wins.

Left-alignment is often desirable for CHAR and CONTROL variables with cells containing long text values. Right-alignment is often desirable for OBSERVATION variables.

See also the statement called ALIGN COLUMN. This statement can be used to align both the label at the top of the column and the values below in the same direction.

Note For variables created with RECODE statements, alignments that are built into the values will take precedence over ALIGN specifications.

Level ALIGN CELLS can be specified for individual columns, specified either by column number or by variable name.

Default

```
ALIGN CELLS CENTER;
```

Example

```
ALIGN CELLS RIGHT;
FOR REPORTS 2 AND 3, VARIABLES CITY, LAST_NAME, COMPANY:
        ALIGN CELLS LEFT;
```

Effect All cells will be aligned to the right, except those for the variables CITY, LAST_NAME and COMPANY in reports 2 and 3. In these reports, the values will be aligned to the left.

ALIGN COLUMN

Format There are three possible alignments for report columns.

```
ALIGN COLUMN LEFT;
ALIGN COLUMN RIGHT;
ALIGN COLUMN CENTER;
```

Meaning Both the labels at the tops of columns and the values entered in the cells below are aligned in the same direction. They can be aligned to the left, right or center. If any label or cell entry is more than one line long, all lines of the label or cell entry will be aligned the same way.

For observation variables, masks and ALIGN COLUMN interact. If both a mask and an ALIGN COLUMN specification apply to the same column containing values for an observation variable, the last specification wins.

ALIGN COLUMN is equivalent to the combination of ALIGN HEAD and ALIGN CELLS where both are specified in the same direction. See these statements for additional details.

If no specific columns are selected with a FOR clause, ALIGN COLUMN applies to all columns, including the special built-in NUMBER column. If you are selecting columns in a FOR clause, you must reference the NUMBER variable by name if you want to include it.

Note For variables created with RECODE statements, alignments that are built into the values will take precedence over ALIGN specifications.

Level ALIGN COLUMN can be specified for individual columns, specified either by column number or by variable name.

Default

```
ALIGN COLUMNS CENTER;
```

Example

```
ALIGN COLUMNS LEFT;
FOR REPORT 1, COLUMNS 2 AND 3:  ALIGN COLUMNS RIGHT;
```

Effect All column labels and the values below will be aligned to the left, except those for columns 2 and 3 of the first report. In these columns, the labels and values will be aligned to the right. For reports other than REPORT 1, the NUMBER column will be aligned to the left.

ALIGN HEAD

Format There are three possible alignments for heading labels.

```
ALIGN HEAD LEFT;
ALIGN HEAD RIGHT;
ALIGN HEAD CENTER;
```

The following are equivalent to the sequence of words ALIGN HEAD. The word LABEL (singular) can be used in place of the word LABELS (plural).

```
ALIGN HEADING LABELS
ALIGN HEAD LABELS
ALIGN HEADING
```

Meaning The labels at the top of each report column can be aligned to the left, right or center. If a heading label is more than one line long, all lines of the heading label will be aligned the same way.

You will find this statement particularly useful if you wish to change the default alignment for all heading labels without entering an alignment specification into each individual label. If you have included an alignment of LEFT, RIGHT or CENTER in an individual label that is used as a heading label, the alignment specified within the label will override the ALIGN HEAD statement.

*******NOTE: For column labels, individual alignment specifications have not yet been implemented in TPL REPORT. *******

See also the statement called ALIGN COLUMN. This statement can be used to align both the label at the top of the column and the data below in the same direction.

Level ALIGN HEAD can be specified for individual columns, specified either by column number or by variable name.

Default

```
ALIGN HEAD CENTER;
```

Example

```
ALIGN HEAD RIGHT;
FOR REPORT 3 VARIABLE CITY:  ALIGN HEAD CENTER;
```

Effect All heading labels will be aligned to the right, except those for the variable CITY in report 3. In this report, the labels above the CITY column will be centered.

ALIGN REPORT

Format There are three possible report alignments.

```
ALIGN REPORT LEFT;
ALIGN REPORT RIGHT;
ALIGN REPORT CENTER;
```

Meaning A report is aligned (LEFT, RIGHT, or CENTER) between the left and right margins. If the report is too wide to fit on the page, it will automatically be divided into as many sections as necessary (one page per section) with each section aligned the same way. If a report is divided into sections by a BANK statement, each section will be aligned the same way.

Level Report alignment can be controlled at the report level. Report alignment cannot change within a report.

Default

```
ALIGN REPORT CENTER;
```

Example

```
ALIGN REPORTS LEFT;
FOR REPORT 3:  ALIGN REPORT CENTER;
```

Effect All reports except the third will be aligned with the left margin on the page. The third report will be centered.

ALIGN TITLE

Format There are three possible alignments for report titles.

```
ALIGN TITLES LEFT;
ALIGN TITLES RIGHT;
ALIGN TITLES CENTER;
```

Meaning The report title can be aligned with the left or right edges of the report, or it can be centered within the width of the report. If the title is more than one line long, all lines of the title will be aligned the same way.

You will find this statement particularly useful if you wish to change the default title alignment for all re[prts without entering an alignment specification into each individual report title. If you have included an alignment of LEFT, RIGHT or CENTER in an individual report title, the ALIGN TITLE specification will not apply to that report title.

Level Title alignment can be controlled at the report level.

Default

```
ALIGN TITLE LEFT;
```

Example

```
ALIGN TITLES CENTER;
FOR REPORT 2:  ALIGN TITLE RIGHT;
```

Effect The report title will be centered for all reports except the second. For the second report, the report title will be aligned with the right edge of the report.

BANK AFTER COLUMN

Format A FOR clause is required to identify the column(s) where banking should take place.

```
FOR [column specification]:  BANK AFTER COLUMN;
```

where **column specification** can be the word COLUMN followed by one or more column numbers and/or the word VARIABLE followed by one or more variable names.

Meaning BANK allows you to specify a break point for a report so that a wide report can be split into sections. Each section is printed on a separate page with all necessary labels repeated for each section of the report.

Level Bank points are specified by column but are controlled at the report level. Bank points cannot change within a report.

Default If the report is too wide for the page, it is banked automatically into as many sections as necessary.

Example

```
FOR COLUMNS 4, 8 AND VARIABLE CITY:  BANK AFTER COLUMNS;
```

Effect Assume that the CITY column is the eleventh column of the report. The first page of the report will contain columns 1-4; the second page of the report will contain columns 5-8; the third page of the report will contain columns 9-11. The remaining columns (assuming there are few enough to fit) will be on the fourth page of the report.

Restrictions If you specify a break point beyond the end of the report, the BANK statement will be ignored. For example, if a report has ten columns, the statement

```
FOR COLUMN 15:  BANK AFTER COLUMN;
```

will be ignored.

FOR VARIABLE **NUMBER**: BANK AFTER COLUMN; does not make sense, because **NUMBER**, if present in the report, is automatically included in each bank of the report.

BANKS PER PAGE

Format BANKS PER PAGE = n;

where n is a number. The word IS can be used in place of =. Both are optional and can be left out altogether.

Meaning If a report is too wide to fit on a page, it is automatically broken into sections called banks. Banking can also be requested explicitly with the BANK AFTER COLUMN statement. By default, each bank begins on a new page. The BANKS PER PAGE statement can be used to print multiple banks on a page.

One line is skipped between banks unless you request a different spacing with the SKIP AFTER BANKS statement. With the statement SKIP 0 LINES AFTER BANKS; the banks will be joined with no space between banks. See the SKIP AFTER BANKS statement for details.

Banks are centered on the page unless a different alignment has been specified.

A report title appears only once on a page regardless of the number of banks. It is formatted according to the most narrow bank on the page and aligned at the left edge of the top bank unless a different alignment has been specified.

Note All banks on a page will take up the same amount of vertical space. If you have different width banks on the same page, you may find that the page ends sooner than you expect, especially if you have a very narrow bank that requires the report title to be broken into several lines. In addition, if one bank has a very long column label that must be broken into several lines, the space requirement for that label will apply to all of the banks. Each bank may then take more vertical space than you expect.

Level BANKS PER PAGE can be specified at the individual report level.

Default BANKS PER PAGE = 1;

Example

```
BANKS PER PAGE = 3;
        FOR VARIABLES COUNTRY AND REGION:
        BANK AFTER COLUMNS;
```

Complete list of shipping records.

Row	A4	BULK_GEN	COM1	COM2	COM3	COUNTRY
1	0000	0	0	04	041	542
2	0000	0	0	04	044	588
3	0000	0	0	08	081	225
4	0000	0	0	08	081	307
5	0000	0	0	08	081	421
6	0000	0	0	08	081	428
7	0000	0	0	08	081	470

Row	Dollars	DPORT	E4	LEG	MONTH	REGION
1	6,519,575	2004	0410	1	12	5
2	448,407	2004	0440	1	12	5
3	239,013	2004	0810	1	12	2
4	2,037,000	2004	0810	1	12	3
5	34,603,211	2004	0810	1	12	4
6	3,415,548	2004	0810	1	12	4
7	5,564,700	2004	0810	1	12	4

Row	Short Tons	TYPE_SERVICE	YEAR
1	54,710	5	85
2	3,662	5	85
3	1,143	5	85
4	9,700	5	85
5	214,295	5	85
6	22,214	5	85
7	28.495	5	85

Example

```
BANKS PER PAGE = 3;
SKIP 3 LINES AFTER BANKS;
```

Effect Each page of the report will contain 3 banks with 3 blank lines between the banks. If the number of banks in the report is not a multiple of 3, then the last page will contain fewer than 3 banks.

Note The **NUMBER** column that appears by default as the first column of a report is repeated for each bank. The row numbers in this column can be used to match up the rows of different banks.

Restrictions There must be enough vertical space on the page for each bank to contain at least one line of data.

CODEPAGE (PROFILE only)

CODEPAGE determines the character set and sort order for your requests and reports. It is most often used for non-English languages that have alphabetic characters not available in the default CODEPAGE.

See also the statement called COUNTRY to set standards for features such as currency symbols and date/time formats.

Important If you add a CODEPAGE or COUNTRY statement to your profile, change a CODEPAGE or COUNTRY statement in your profile, or make changes to **country.tpl**, ***you need to restart TPL*** to activate the changes. When running a report request, you must use the same CODEPAGE and COUNTRY statements that you used when processing your codebook. Otherwise, you will have conflicting standards. In particular, conflicts in CODEPAGE will cause the sort order to be scrambled.

Note If you use CODEPAGE and/or COUNTRY statements, place them at the beginning of your profile.

Format

```
CODEPAGE = cp-name;
```

where **cp-name** is a CODEPAGE name. The word IS can be used in place of =. Both are optional and can be left out altogether.

Level CODEPAGE applies to all reports.

Windows Default CODEPAGE = WIN88591;

UNIX Default CODEPAGE = ISO88591;

Meaning The CODEPAGE statement determines the following:

1. the character set that includes the alphabet you wish to use in names and labels,
2. the character set for PostScript reports, and
3. the sort sequence for the character set.

The character sets associated with different CODEPAGES are contained in files that are installed in the TPL system directory. These files have names that end with **.cp**. The names correspond to the supported codepages. For example, for CODEPAGE = WIN88591; the character set information is contained in the file **WIN88591.cp**. For CODEPAGE = ISO88591; the character set information in contained in the file **ISO88591.cp**.

The **.cp** files are ASCII text files that can be printed or displayed on the screen but you should not change them. Please tell us if you think that a change should be made for your alphabet.

Alphabet for Names

The TPL alphabet depends on the CODEPAGE. The default CODEPAGE is adequate for many, but not all, languages. If you need additional letters, look at the CODEPAGES in the Appendix to find an alphabet that you can use.

The Character Set for Printing PostScript

TPL automatically provides the printable characters for the selected CODEPAGE.

The Sort Sequence

The proper order for sorting depends on the character set used. TPL will use the sequence that goes with the character set selected by the CODEPAGE statement.

The sort sequences for all character sets are stored in a file called **sort.tpl** that is installed in the TPL system directory. This is an ASCII text file that you can print or display on the screen. It can be edited according to the instructions included at the beginning of the file. We would appreciate your telling us if you think that changes should be made in the **sort.tpl** file that we distribute with the TPL software.

If You Need to Select a CODEPAGE

Consult the character set tables in the Appendix called "Character Sets" and use the codepage name from any table that contains the characters that you need.

COLOR Defaults

Format

```
DEFAULT COLOR = r g b;
LABEL COLOR = r g b;
RULE COLOR = r g b;
SYMBOL COLOR = r g b;
```

where **r**, **g** and **b**, are numbers between 0 and 100 (inclusive) which specify **r**ed, **g**reen, and **b**lue components of color.

Meaning

COLOR defaults can be set for all characters and rules (lines) used in reports. The defaults are applied as follows.

DEFAULT COLOR is the print color for the entire report if no other colors are specified. If RULE COLOR and LABEL COLOR are specified, the DEFAULT COLOR remains as the default color for report cells.

RULE COLOR is the print color for rules. It applies to all rules, including rules added by the FORMAT statement **RULE EACH n**. If no explicit RULE COLOR is specified, rules are printed in the default color.

LABEL COLOR is the print color for all text in reports except character strings in cell masks. These strings are printed in the default color. If no explicit LABEL COLOR is specified, all labels, titles and footnote texts are printed in the default color.

SYMBOL COLOR is the print color for all footnote symbols. If SYMBOL COLOR is not set explicitly, the default label color is used for symbols.

In cases where color specifications are entered directly into ***individual*** report elements such as labels, masks or footnotes, these individual specifications will take precedence over the ***default*** COLOR specifications.

COLOR defaults apply only to characters and rules. For background shading in color or grey, see the SHADE statement (or COLOR shading).*****NOTE: Shading has not yet been implemented for TPL REPORT. *****

Note on Cell Color

There is no default color statement that applies only to report cells. If you wish to change the cell color without affecting other report elements that are to be displayed in the default color, you can use the FORMAT statement, REPLACE CELL COLOR (or the equivalent REPLACE MASK COLOR), to replace the mask color for entire reports. This statement affects only cell color and does not change the other mask characteristics of a cell.

Note on Underlining

The color for underlining is determined by the labels or masks to which the underlining applies if it is part of the font for the labels or masks.

Level All COLOR statements can be specified for individual reports.

Default The default color is black.

Example

```
DEFAULT COLOR = 0 20 99;
FOR REPORT 1:  REPLACE TITLE WITH COLOR 100 0 0 'Red report title';
FOR REPORT 2:  RULE COLOR = 100 0 0;
```

Effect All reports will be printed in the default color 0 20 99 (a shade of blue) except as follows. The first report will have a title in the color 100 0 0 (red). The rules in the second report will be printed in the color 100 0 0 (red). The rest of the second report will be printed in the default color.

Alternate Format for the COLOR Statements

Colors can also be referenced by name where the colors have been defined in a file called **color.tpl**.

Format

```
DEFAULT COLOR = color;
LABEL COLOR = color;
RULE COLOR = color;
SYMBOL COLOR = color;
```

where **color** is a user-selected name that has been assigned to a color definition in the **color.tpl** file.

The **color.tpl** file is installed as part of the TPL system with several colors already defined. You can customize this file to add the colors of your choice. For complete details, see the section called "General Information about Color" in the Color chapter.

Example

```
DEFAULT COLOR = BLUE;
FOR REPORT 2:  RULE COLOR = BROWN;
FOR REPORT 2, COLUMN 1:  REPLACE MASK WITH COLOR RED 999.9;
```

Effect All reports will be printed in the default color BLUE except the second report. The rules in the second report will be printed in the color BROWN. Since the color RED is included within the mask for the first column of the second report, the data values in that column will be printed in RED. The rest of the second report, the labels and the other data values, will be printed in the default color.

COLOR = NO

In some cases a report is designed to be printed on a color PostScript printer but must be previewed on a monochrome PostScript printer. Colors are printed on a monochrome printer as shades of grey. The resulting report is often hard to read, and different colors sometimes convert to the same shade of grey. Two additional statements may be added to your FORMAT request to deal with this.

Format

```
COLOR = NO;
REPLACE COLOR c WITH FONT f;
```

where **c** is a color name or **r g b** specification and **f** is a font name and optional font size.

COLOR = YES; is the default. If you have not specified COLOR = NO; the REPLACE COLOR statements are ignored.

If COLOR = NO; is selected, and no REPLACE COLOR statement is provided, all color information is ignored and the reports are printed in black and white.

If COLOR = NO and REPLACE COLOR statements are included, the reports are still printed in black and white but the REPLACE COLOR statements are used to substitute special fonts for color.

Note Replacement of table elements such as masks or labels in statements that follow a REPLACE COLOR statement may nullify the color replacement for those table elements. You can avoid this possibility by putting the REPLACE COLOR statement at the end of the format request. For example:

```
FOR REPORTS ALL:  REPLACE COLOR GREEN WITH FONT HB;
```

Example Suppose you have a report intended for printing on a color printer and suppose that you wish to use a RED data mask to emphasize certain report cells. You need to preview this report on a monochrome printer but you want the red cells to stand out. You can accomplish this by adding the following two statements to your FORMAT request or profile.

```
COLOR = NO;
REPLACE COLOR RED WITH FONT HB;
```

Now all red cells will be displayed in Helvetica Bold. When you are ready to print on your color printer, just replace COLOR = NO; with COLOR = YES; The REPLACE statements need not be removed since they will be ignored.

Restrictions Changing the color of a string of characters does not change its length, but changing the font and especially the font size of the string does change its length. REPLACE COLOR statements are intended for use in previewing reports before final printing on a color printer. To aid in doing this, all report layout is done before the font changes are applied. Consequently, if a wider font is specified in a REPLACE COLOR statement, some character strings may be too wide to fit in their space. Thus, they may overlay some of the adjacent rules or stub filler dots. If the font size is not changed, this will rarely happen but it is possible.

Alternate Approach

If you want to produce a publication-quality report that emphasizes certain report cells or labels with color and can be printed on both a monochrome and color printer, you can assign both a color and a distinguishable font to the cells or labels that you want to emphasize. Do not use a REPLACE COLOR statement but do use COLOR = NO; when printing on a monochrome printer. On a color printer, both the color change and the font change will highlight the emphasized cells or labels. On a monochrome printer, the font change will show the emphasis.

COLUMN WIDTH

Format

```
COLUMN WIDTH = amount  [ unit ];
```

where **amount** is a number and **unit** is optional. If no unit is specified, characters are assumed. If a unit is specified, the amount can be a decimal number and unit can be expressed as **inches**, **cm** or **points**.

The word **IS** can be used in place of **=**. Both are optional and can be left out.

Meaning

For specified columns, make the report columns **n** characters (or other units) wide. The column width includes the column divide character. If the column width is **n** characters, then the column will have one less than **n** character positions for data plus one position for the divide character.

You can specify different column widths for different columns, referencing the columns by column number or by variable. If the column width action is not restricted by a FOR clause, the column width will be the same for all columns in all reports.

See also COLUMN WIDTH AUTOMATIC in this chapter for automatic adjustment of columns widths to fill the available space.

Level

Column width can be specified for individual columns or variables but is controlled at the report level. For a particular column, the column width will be the same throughout any one report.

Default

The default width for each column is based on the data type. In all cases, the width includes one extra character to allow for a blank space between columns. See the REPORT chapter for additional details.

For the NUMBER variable, the column width is 10.

Example

```
COLUMN WIDTH = 15;
FOR VARIABLE STATE_OF_RESIDENCE:  COLUMN WIDTH = 12;
FOR REPORT 3 COLUMNS 2 AND 4:  COLUMN WIDTH = 8;
```

Effect

The first FORMAT statement sets column width to 15 for all reports. If the second statement follows in the same format request, it sets column width to 12 for any STATE_OF_RESIDENCE columns. The third statement sets column width at 8 for columns 2 and 4.

Restrictions

In line printer mode, the minimum width is 3; in PostScript mode, it is 4.

COLUMN WIDTH AUTOMATIC

Format

```
COLUMN WIDTH = AUTOMATIC;
COLUMN WIDTH = AUTOMATIC  MAXIMUM = n [ unit ];
```

where **n** is a number that specifies a width, and **unit** is optional. If no unit is specified, characters are assumed. If a unit is specified, the amount can be a decimal number and unit can be expressed as **inches**, **cm** or **points**.

The word IS can be used in place of =. Both are optional and can be left out altogether. AUTO is a synonym for AUTOMATIC and MAX is a synonym for MAXIMUM.

Meaning You can use these statements if you wish to "stretch" reports to the full width of the page minus the margins; if you wish to have all reports be the same width, regardless of the number of columns; or if you wish to have all banks of a report be the same width, even if they have different numbers of columns.

If you specify COLUMN WIDTH AUTOMATIC; extra space is added to the columns so that the report takes up the full width of the page minus the left and right margins.

In some cases, COLUMN WIDTH AUTOMATIC; will make columns wider than is desirable. This is especially likely when a banked report has only a small number of columns in the last bank. COLUMN WIDTH AUTOMATIC MAXIMUM n; can be used to limit the effect of automatic column widths so that no column will have a width greater than n. Note that if you have set an explicit width for a column that is wider than the maximum automatic value, this column will not have its width reduced. Note also that if you specify a maximum automatic value for columns, the final width of the report may not be as wide as the page width minus the margins.

Level These statements can be specified at the individual report level.

Default

```
COLUMN WIDTH = 10;
```

unless explicitly specified otherwise.

Example Assume that for a report with 5 columns, we have the following FORMAT statements:

```
PAGE WIDTH = 80;
LEFT MARGIN = 10;
RIGHT MARGIN = 10;
COLUMN WIDTH AUTOMATIC;
```

Effect The available space for the columns will be 80 - 10 - 10 = 60 (i.e. the page width minus the margins). Each column will be expanded from the default width of 10 to a width of 12 so that the full page width will be used.

Restrictions The NUMBER column is not affected by this statement.

These statements can work with greater precision in PostScript mode than in line printer mode. For example, if COLUMN WIDTH AUTOMATIC; is specified in line printer mode and the available space divided by the number of columns yields a fractional number, some adjustments must be made to allow for the fact that all characters and spaces have the same width.

If these statements are used with PAGE WIDTH AUTOMATIC; they will be ignored, regardless of the order of the statements.

COUNTRY (PROFILE only)

Note If you are using U.S. standards for such things as data formats, currency symbol and date/time formats, you do not need to use this statement. **If you do use a COUNTRY statement, place it at the beginning of your profile.** Before doing this, you should check the profile to see if any decimal numbers are used. If they are, you may need to edit them so that they conform to your country's standards as described below in the section called "Separators in Masks and Decimal Constants".

See also the statement called CODEPAGE to select a character set and sort order for languages other than English.

Format

```
COUNTRY = country-name;
```

where **country-name** is the name of a country listed in the **country.tpl** file. The word IS can be used in place of =. Both are optional and can be left out altogether.

Meaning The COUNTRY statement lets you select appropriate standards for items such as thousand separator, decimal separator, currency symbol, and date/time formats. The standards are set in the file called **country.tpl**. This file is installed in the TPL system directory. It is an ASCII file that can be printed or displayed on the screen. The **country-name** that you enter in the COUNTRY statement must match a name in **country.tpl**. Associated with each country in the file, you can see the standards that have been set for that country.

The countries that currently have entries in the file are listed below. If your country name is not on this list, look in **country.tpl** to see if it has been added. If your country is not in **country.tpl**, you will need to edit the file to add the standards for your country. If you add a country to **country.tpl**, please send your entry to us. We will add it to our **country.tpl** file so that your country will always be included when you get new versions of TPL software.

*******NOTE: The special treatment for currency symbols has not yet been implemented for TPL REPORT, but currency symbols can be entered as strings in masks. See the chapter called "Masks". *******

AUSTRALIA	JAPAN
AUSTRIA	MEXICO
BELGIUM_DUTCH	NETHERLANDS
BELGIUM_FRENCH	NZEALAND
BRAZIL	NORWAY
CANADA	POLSKA
CANADA_FRENCH	PORTUGAL
DENMARK	SKOREA
FINLAND	SPAIN
FRANCE	SWEDEN
GERMANY	SWITZERLAND
ICELAND	TAIWAN
IRELAND	UK
ITALY	US

If there is an entry for your country but you wish to change the standards, you can edit the **country.tpl** file. Follow the directions contained in the file accurately. Errors in this file could cause your TPL jobs to fail. As with **profile.tpl**, you can edit the file in the TPL system directory, or you can make a customized copy to use in the directory where you are running your TPL jobs.

Separators in Masks and Decimal Constants

For the default country US, the decimal separator is "." and the thousands separator is "," . The COUNTRY statement lets you choose different defaults so that you can enter masks and numeric constants and have values print with the separators that are customary in your country.

Enter **masks** according to your country standard. For example, if your thousands separator is "." and your decimal separator is "," then a valid mask would be **MASK 9.999,99** .

Note that if your country uses a period "." as the decimal separator you must enter comma "," as the thousands separator in masks. In the output, the data will use the correct symbol for the thousands separator if it differs from comma. Similarly, if your country uses a comma "," as the decimal separator, you must enter a period as the thousands separator in masks.

For example, some countries, such as France and Finland, use comma "," as the decimal separator and blank as the thousands separator. Blank can be entered in **country.tpl**, but a mask with a blank will cause a syntax error. If you use blank as the thousands separator, you must enter the mask using a period as the thousands separator. The data values will be printed correctly with blank as the thousands separator.

Enter **numeric constants** according to your country standard. For example, if your decimal separator is "," then **457,22** is a valid entry. The thousands separator is not relevant, because thousands separators cannot be entered in numeric constansts.

Note This feature does not extend to observation variable values in a data file. If they have decimal or thousands separators, they must conform to US standards, i.e. period for decimal and comma for thousands.

Effect on Currency Formats

If you have entered the COUNTRY statement in your profile, the currency symbol will be taken from the corresponding country entry in **country.tpl**. This entry also determines whether the symbol should be displayed before or after the money value, with or without a blank between. As already noted, you can edit **country.tpl** if you wish to change the symbol or its placement.

The rules for entering currency symbols in masks in codebooks and statements vary depending on the type of currency symbol for your country. In any case, if your country name and codepage specifications are correct, a Postscript output will have the correct currency symbol inserted in the correct place in your data values. If you are using a non-Postscript printer, what will be printed depends on your printer. Changes to the **country.tpl** file can usually fix non-Postscript printer problems.

1. For some countries, the currency symbol is a single special character such as the UK Sterling symbol. For this type of currency symbol, you may enter either $ or the special symbol in your print masks.
2. For other countries such as France, the currency symbol is a regular letter in the alphabet. In such cases you must use the US $ in your print masks, but the letter will be used as the currency symbol in formatting data values.
3. For currency symbols such as Kr and Cr$ that contain more than one character, you must use the US $ in your print masks, but the correct combination of symbols will be used in formatting the data.

Examples

```
COUNTRY = Denmark;
User specified mask is: $999,99
Output for 332.76 is:  332,76Kr

COUNTRY = UK;
User specfied mask is: £999.99
Output for 332.76 is: £332.76

COUNTRY = CANADA_FRENCH;
User specfied mask is: $99.999,99
Output for 54332.76 is: $54 332,76
```

In some countries it is customary to insert a blank space between the currency symbol and the value. Because space is often limited in columns of data, we have not included blank characters for currency symbols in our **country.tpl** file. If you wish to have the blank space inserted, you can change the currency style field in your **country.tpl** file to request the blank space for your country's entry.

Note TPL software does not support currencies that put the currency symbol at the decimal point place. Please contact us if you have a requirement for this format.

Special Treatment for Currency Symbols in Output

TPL provides special treatment for masks that contain the US currency symbol $. For example, in a column of numbers with a mask of **$999.99**, only the first number in the column will be displayed with a $. (See the MASK chapter for a more detailed description of the $ treatment.) Non-US currency symbols are given similar treatment.

Date and Time Formats

Whenever date and time are displayed by TPL, the formats are determined by COUNTRY. For example, if you use DATE or TIME in the FORMAT statement called PAGE MARKER and have specified **COUNTRY = SWITZERLAND**, the statement:

```
PAGE MARKER = "Page " NUMBER "  Job run on " DATE " at " TIME;
```

will produce reports with page numbers, date and time in the following format:

```
Page 1  Job run on 31.12.95 at 14,24,38
```

Level COUNTRY applies to all reports.

Default

```
COUNTRY = US;
```

Restrictions When running a report request, you must use the same CODEPAGE and COUNTRY statements that you used when processing your codebook. Otherwise, you will have conflicting standards. In particular, conflicts in CODEPAGE will cause the sort order to be scrambled.

CSV DIVIDER

Format CSV DIVIDER = "c";

where **c** can be any character, including blank, and the quotes can be single or double. The word IS can be used in place of =. Both are optional and can be left out altogether.

Meaning For exported CSV files, the default divider (delimiter) between values is comma. CSV DIVIDER can be used to specify a different divider character.

Level The divider is controlled at the request level. The same divider applies to all reports within the same report request.

Default CSV DIVIDER = ",";

Example CSV DIVIDER = " ";

Effect When the reports are exported to CSV format, the values will be separated by a blank character.

Restriction If a Tab is entered as the divider, it will be treated the same as a blank. If you are using the Windows version of TPL and do the export interactively from Ted, you can select Tab as the divider.

CSV OUTPUT (UNIX only)

Format CSV OUTPUT = YES or NO or PROMPT;

Normally, when you have created PostScript reports, TPL REPORT will prompt you at the end of a job to find out if you would like to export the reports to other formats. To prevent the prompt for CSV, you can use this statement with **YES** or **NO**.

DATA REPORT

Format

```
DATA REPORT;
DATA REPORT ZERO FILL;
```

Meaning The DATA REPORT statement is especially useful if you are extracting data for subsequent use with other software such as TPL TABLES, spread sheets or graphics programs.

It formats the reports so that they can be used as data files. All title and label information is removed along with the space between pages. As with normal reports, each data report is put into a separate file. Each column, including the last, has a single blank space at the right end.

Only masks and cell alignments remain in effect. If you need decimal points in the numbers or if you plan to use the data reports as input to software that does not allow commas in the data, you may need to add your own mask for the data since the default mask inserts commas in numbers that are longer than 3 digits and does not include decimal points. Since the default alignment for values is CENTER, you may also wish to use masks or ALIGN CELL statements to left or right-align the data according to the requirements of the software you plan to use with the data reports.

Any other display information that you want to keep in the tables can be preserved with RETAIN statements.

Note If you have data values wider than the default column widths, you will probably need to add COLUMN WIDTH specifications so that the values won't "wrap" and add unwanted lines to the data reports.

The page width of a data report is automatically calculated to hold all columns of the report so that all values in a data row will be on the same line if the column widths are adequately large. In other words, wide data reports are not automatically banked.

ZERO FILL

By default, any space not occupied by the values or associated mask characters will be filled with blanks. If you use the ZERO FILL option, these blanks will be replaced with zeros.

Level Data reports can be specified at the individual report level.

Default Reports are formatted with complete title, label and footnote information, margins and pagination.

Example

```
FOR REPORT 1:  ALIGN COLUMNS RIGHT;
               DATA REPORT;
```

Effect Format the first report as a data file with the values right-adjusted within each column.

Example

```
REPLACE MASK WITH 9999;
ALIGN COLUMNS RIGHT;
DATA REPORTS ZERO FILL;
```

Effect Format all reports as data files with the data values right-adjusted within each column and no commas or decimal places in values of observation variables. Replace blanks to the left or right of data values with zeros.

Note The DATA REPORTS statement is equivalent to the following collection of FORMAT statements:

```
TOP MARGIN = 0;
LEFT MARGIN = 0;
PAGE LENGTH = AUTOMATIC;
PAGE WIDTH = AUTOMATIC;
DELETE TITLE;
DELETE HEADING;
DELETE FOOTNOTES ALL;
DELETE ALL RULES;
RETAIN BLANKS;
PAGE MARKER = '';
```

You must take care that you do not unintentionally override parts of the DATA REPORT effect by following the DATA REPORTS statement with a FORMAT statement that conflicts with one of the above. For example, the sequence

```
DATA REPORTS;
PAGE LENGTH = 66;
```

would override the PAGE LENGTH AUTOMATIC statement that is built into DATA REPORTS and cause gaps to appear between pages.

Restrictions DATA REPORTS cannot be used in POSTSCRIPT mode.

Special indicators, such as the (c) that indicates a computation error, are not removed from data reports.

DELETE

The following statements are defaults. They are described with their corresponding RETAIN statements. See the RETAIN statements for details.

```
DELETE ALL RULES;
DELETE BLANKS;  (default for CHAR values formatted for printing)
DELETE CROSS RULES;
DELETE DOWN RULES;
DELETE SIDE RULES;
```

DELETE COLUMNS

Format DELETE COLUMNS;

Meaning DELETE COLUMNS usually only makes sense if used with a FOR clause that restricts the number of columns deleted. The format of the report is adjusted so that it looks as if the deleted columns were never there.

In the FOR clause, columns can be selected by column number or by variable name.

If the **NUMBER** column is present, *it can be referenced only by name* (FOR VARIABLE NUMBER ...). If the NUMBER column is in its default position as the left-most column, then the next column after it is COLUMN 1.

Level Deletion of columns can be specified for individual columns or variables.

Default RETAIN COLUMNS;

Example FOR REPORT 3 COLUMN 1: DELETE COLUMN;

Effect The first column of the third report will be deleted. The format of the report will be adjusted accordingly. If the NUMBER column is the left-most column, then the deleted column will be the one following the NUMBER column.

Example FOR VARIABLES NUMBER AND REGION, COLUMN 3: DELETE COLUMNS;

Effect For all reports, the NUMBER and REGION columns will be deleted, along with column 3. If the NUMBER column is on the left, column 3 is the third column following the NUMBER column.

DELETE HEADING

Format

```
DELETE HEADING;          or
DELETE HEADER;           or
DELETE HEAD;
```

Meaning The column headings labels are removed from the report. The report title is immediately followed by the first line of data.

Level Heading deletion can be specified for individual reports.

Default

```
RETAIN HEADING;
```

Example

```
FOR REPORT 1: DELETE HEADING;
```

Effect The column headings will be removed from the first report in the request. All of the other reports will be formatted with the column heading labels present.

DELETE LEADING ZEROS

Format DELETE LEADING ZEROS;

Meaning When decimal values less than zero are printed, no leading zeros are printed to the left of the decimal point. For example, the number 0.45 will print as .45 with no zero to the left of the decimal point.

Level Deletion of leading zeros can be specified for individual reports.

Default RETAIN LEADING ZEROS;

Example DELETE LEADING ZEROS;

Effect

```
 .16  2.05
2.05   .16
 .53   .53
 .94   .94
```

DELETE REPORT

Format DELETE REPORT;

Meaning If not used with a FOR clause, DELETE REPORTS will cause all reports to be deleted from the output. If it is used with a FOR clause that specifies which reports should be deleted, only the specified reports will be deleted from the report output. This statement can be useful if you have a report request for many tables, but you want to create only certain ones at any particular time.

Note DELETE REPORT does not actually delete reports. Instead, it prevents them from being created. *The statement must be present when the job is first run*; otherwise, it will have no effect. You cannot use this statement to delete reports after they are created.

Level Report deletion can be controlled at the individual report level.

Default

```
RETAIN REPORTS;
```

Example

```
FOR REPORTS 2 AND 3:  DELETE REPORTS;
```

Effect The second and third reports will be deleted. All others will be retained.

Example

```
DATA REPORTS;
FOR REPORTS ALL:  DELETE REPORTS;
FOR REPORT 3:  RETAIN REPORT;
```

Effect The third report will be formatted as a data file. All other reports will be deleted.

DELETE TITLE

Format DELETE TITLE;

Meaning Reports are formatted without title lines at the top of each page.

Level Title deletion can be controlled at the report level.

Default RETAIN TITLE;

Example FOR REPORT 2: DELETE TITLE;

Effect The second report will be formatted without title lines at the top of each page.

DISPLAY NAME (UNIX/Linux Profile only)

Windows Note The Windows version of TPL REPORT uses TED, the TPL editor, to display Postscript reports. **DISPLAY NAME** is ignored.

Format DISPLAY NAME = *PostsScript-displayer;*

where *PostScript-displayer* can be either just the program name or the name including a full path if needed.

Meaning When a TPL REPORT job run in PostScript mode completes successfully, you will be asked if you wish to display the report(s). If you answer "yes", the report(s) will be displayed using the *PostScript-displayer* as a separate process. If you have multiple reports, each will be opened in a separate process.

Examples

Sun Solaris:

```
DISPLAY NAME = pageview;
DISPLAY NAME = /usr/openwin/bin/pageview;
```

Linux (using KDE):

```
DISPLAY NAME = kghostview;
```

DOWN RULE WEIGHT

Format DOWN RULE WEIGHT = n;

where **n** is a number. The word IS can be used in place of =. Both are optional and can be left out altogether. The word LINE can be used in place of the word RULE.

Meaning This statement applies in PostScript mode only. If RETAIN DOWN RULES is specified to retain vertical lines between the columns of a report, the DOWN RULE WEIGHT statement can be used to adjust the thickness of these lines. These lines are called down rules. The rule weight is expressed in points where each point is 1/72 inches.

The thickness of the rules will increase or decrease according to the rule weight number specified. Note that the appearance for a particular rule weight on the printed page will vary from printer to printer. This is especially true with printers of different dpi (dots per inch).

DOWN RULE WEIGHT can be restricted to specific columns. This is useful, for example, if you wish to emphasize the division between certain columns by increasing the width of the dividing rule. In the FOR clause, columns can be selected by column number or by variable name. When a column is selected, the new rule weight is applied to the dividing rule that follows the specified column.

Note If the NUMBER column is the left-most column (the default position for it), it does not have a column number. This means that the column numbering starts at the first column following the NUMBER column on the left. In this case, the NUMBER column can only be referenced by name: FOR VARIABLE NUMBER.

This statement does not apply to side rules. A weight can be specified for side rules as part of the RETAIN SIDE RULES statement.

See also, the statements RULE WEIGHT and RETAIN SIDE RULES WEIGHT = n for additional ways to control the thickness of lines in reports.

Level DOWN RULE WEIGHT can be specified at the individual column level.

Default DOWN RULE WEIGHT = .5;

unless the default rule weight has been set to a different value with a RULE WEIGHT statement.

Example

```
FOR REPORT 3:  RETAIN DOWN RULES;
FOR REPORT 3 COLUMNS 2, 4:  DOWN RULE WEIGHT = 1.5;
```

Effect All rules in the third report will have the default rule weight except the rules that follow columns 2 and 4. These two rules will be thicker than the others.

Example

```
RETAIN DOWN RULES;
FOR VARIABLE NUMBER:  DOWN RULE WEIGHT = 2;
```

Effect The rule between the NUMBER column and the column that follows it will be thicker than the rules between the other columns.

Restrictions The rule weight value must be greater than or equal to zero. A rule weight of 0 does not make the rule disappear. Instead, it results in the thinnest rule that is possible on your PostScript output device.

If the rule weight value is too large, the rules will be so thick that they will make broad bands that overlay the columns. This is usually undesirable. For example, the statement:

```
DOWN RULE WEIGHT = 72;
```

will create broad black bands that are 1 inch wide (1 pt. = 1/72", so 72 points = 1").

EDITOR (UNIX Profile only)

Windows The Windows version of TPL REPORT is linked to TED, the TPL editor. EDITOR statements have no effect.

UNIX There are two EDITOR statements. They are used only in **profile.tpl** and are initially set at installation time if you have indicated that you would like to have TPL REPORT linked to your editor. They are described below in case you need to change or add them after installation.

Format

```
EDITOR NAME = editor_name;
EDITOR FILE = editor_file;
```

Meaning TPL REPORT has been designed so that you can use the text editor of your choice to create codebooks, report requests and format requests. Any editor that creates stand-alone ASCII text files is acceptable.

If you choose to link TPL REPORT to your editor, TPL REPORT will automatically transfer to your editor when a job stops because of an error.

Editor Name

The **editor_name** should be the name you use to start your editor. For example, if you start your editor by entering **ED**, the editor name statement in your profile should be:

```
EDITOR NAME = ED;
```

Path names are allowed but not required.

Editor File

TPL REPORT assumes that you can start your editor with a command that includes the name of the file to be edited. TPL REPORT uses a file name of **TPLTEMP** when it transfers to your editor, so the EDITOR FILE statement is:

```
EDITOR FILE = TPLTEMP;
```

Some editors require a special extension such as **DOC** or **TXT** for any file to be edited. If this is the case with your editor, include the required extension in your EDITOR FILE statement. For example, if the required extension is **TXT**, use the statement

```
EDITOR FILE = TPLTEMP.TXT;
```

EPS OUTPUT (UNIX only)

Format EPS OUTPUT = YES or NO or PROMPT;

Normally, when you have created PostScript reports, TPL REPORT will prompt you at the end of a job to find out if you would like to export the reports to other formats. To prevent the prompt for EPS, you can use this statement with **YES** or **NO**.

EXTRA LEADING

Format EXTRA LEADING = n;

where **n** is a number. The word IS can be used in place of =. Both are optional and can be left out altogether.

Meaning This statement applies in PostScript mode only. It can be used to regulate the amount of space between lines in reports.

Leading (rhymes with "heading") is the space between lines of text or data. In PostScript mode, TPL REPORT automatically adjusts this space in proportion to the font size you have chosen. If this amount is not appropriate for your reports, you can change the spacing with the EXTRA LEADING statement. Increasing the leading number will increase the amount of space between lines; decreasing the leading number will decrease the amount of space between lines.

A value of 0 for EXTRA LEADING will give the PostScript default spacing. Since the default PostScript spacing is often too close for reports, TPL REPORT uses an extra leading value of .15 to increase the spacing by a small amount.

The PostScript font sizes include the leading. For each line, most of the vertical space will be occupied by the printed characters and part will be reserved for the space between lines. For example, if the font size is 12 (points), each line, including the leading, will take 12/72 = 1/6 inch of vertical space. Thus, there will be 6 lines per inch. (A point is 1/72 inch).

In the EXTRA LEADING statement, the extra leading value is multiplied by the font size to determine the extra amount of spacing. For example, **EXTRA LEAD-**

ING = .5; will add .5 * font size to line spacing. If the font size is 12, the extra leading will be .5 * 12 points or 6 points, and each line will take 12 + 6 = 18 points of vertical space. 18 points = 1/4 ", so there will be 4 lines per inch. The characters will be the standard PostScript character size for a font size of 12, but there will be much more space between the lines.

If there is more than one font size specification for a line, the extra leading calculation will be based on the largest font size for the line.

Level Extra leading can be controlled at the individual report level.

Default EXTRA LEADING = .15;

Example FOR REPORT EMP_RPT: EXTRA LEADING = .2;

Effect For the report named EMP_RPT, line spacing will be increased beyond the standard PostScript leading by .2 * font size. If the font size for a line is 10, the extra space will be 2 points.

Restrictions The EXTRA LEADING value cannot be less than 0.

FONT

Print style and size can be specified with the FONT statement.

Note The FONT statement is only effective in PostScript mode. It is ignored otherwise.

Format

```
report-element FONT = fontname  fontsize;
```

where **fontname** is the TPL REPORT abbreviation for the PostScript font name, and **fontsize** is a number.

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether. **Fontsize** is optional for all except the DEFAULT FONT statement.

Individual masks and labels, including titles and page markers, can contain FONT specifications that will over-ride the FONT statements. For more information, see the chapters on Masks and Labels.

Example

```
TITLE FONT = HB 10;
```

Effect The title font will be 10 pt Helvetica Bold.

Level The DEFAULT font applies to the entire request, but fonts for other report elements can be specified at the report level.

Report Elements

Fonts can be specified for the following report-elements. Note that there is no font specification that applies to report cells only. To change the mask font only for observation variables, see the statement REPLACE MASK FONT.

DEFAULT	The DEFAULT font applies to the report cells and any report elements not otherwise specified.
TITLE	
TITLE CONTINUATION	
CONDITION LABELS	
CONDITION LABELS IN HEADING	
VARIABLE LABELS	
VARIABLE LABELS IN HEADING	

Font Names

You can choose from any of the following fonts. Refer to them by TPL REPORT abbreviation. For example,

```
TITLE FONT TBI;
```

specifies **T**imes-**B**old**I**talic.

Abbreviations	PostScript font names
C	Courier
CB	Courier-Bold
CI	Courier-Oblique
CBI	Courier-BoldOblique
T	Times-Roman
TB	Times-Bold
TI	Times-Italic
TBI	Times-BoldItalic
H	Helvetica
HB	Helvetica-Bold
HI	Helvetica-Oblique
HBI	Helvetica-BoldOblique
N	Helvetica-Narrow
NB	Helvetica-Narrow-Bold
NI	Helvetica-Narrow-Oblique
NBI	Helvetica-Narrow-BoldOblique
A	AvantGarde-Book
AB	AvantGarde-Demi
AI	AvantGarde-BookOblique
ABI	AvantGarde-DemiOblique
B	Bookman-Light
BB	Bookman-Demi
BI	Bookman-LightItalic
BBI	Bookman-DemiItalic
S	NewCenturySchlbk-Roman
SB	NewCenturySchlbk-Bold
SI	NewCenturySchlbk-Italic
SBI	NewCenturySchlbk-BoldItalic
P	Palatino-Roman
PB	Palatino-Bold
PI	Palatino-Italic
PBI	Palatino-BoldItalic

Z	ZapfChancery-MediumItalic
D	ZapfDingbats
Y	Symbol

Font Sizes

Font sizes are specified in points. A point is 1/72", so there are 72 points in one inch. The font size includes the space between the lines.

Examples If the font size is 12, each line takes 12 points of vertical space or 12/72 = 1/6". With font size 12, there are 6 lines per inch.

> If the font size is 8, each line takes 8 points of vertical space or 8/72 = 1/9". With font size 8, there are 9 lines per inch.

General Rule To get larger characters in your report, increase the font size; to get smaller characters, decrease the font size.

A font size must be specified for the DEFAULT FONT. For all other fonts, if no size is specified, it will be whatever size is already in effect for that report element. For example, if the title font is set at H 12 (Helvetica 12) in the profile and the statement **TITLE FONT TBI;** (Times-BoldItalic) is included in the format request, the title will be printed in Times-BoldItalic with a size of 12.

Adding Underline to Fonts

You can add underlining to any of the PostScript fonts by adding a **U** to the font specification. In the following example, the default font is set to **HU** for **H**elvetica **U**nderline, the title font is set to **HBIU** for **H**elvetica **B**old **I**talic **U**nderline, and the variable label font is set to the non-underlined **H**elvetica font **H.**

```
Default font = HU 10;
Title font = HBIU 12;
Variable label font = H 10;
```

Report on the Status of Selected Loans

Row	Loan Number	Classification	Balance	Month Maturing
1	41	1 to 4 Family	$2,594	06
2	75	1 to 4 Family	$11,135	03
3	106	1 to 4 Family	$3,921	00
4	786	1 to 4 Family	$4,059	04
5	7112	1 to 4 Family	$20,448	08
6	7574	1 to 4 Family	$42,323	06
7	9152	1 to 4 Family	$28,500	12
8	9210	1 to 4 Family	$145,500	06
9	9222	1 to 4 Family	$106,693	06
10	9306	1 to 4 Family	$33,746	01

To insert horizontal lines between rows, spanning the entire width of the report rather than underlining only the cell values, see the FORMAT statement RULE EVERY.

Using the Symbol and Zapf Dingbats Fonts

The character sets for the Symbol and ZapfDingbats fonts are shown in the Appendix. As you will see if you look at these character tables, the Symbol font includes numbers but the ZapfDingbats font does not. Neither of these fonts can be used to print alphabetic characters.

Since the Symbol and ZapfDingbats fonts do not contain the usual alphabetic characters, these fonts would not normally be used in FONT statements To use characters from these fonts in labels, add the FONT specifications to individual labels as described in the **Labels** chapter.

Although most of the characters in these special fonts are not on your keyboard, you can enter them in labels by typing **\nnn** where **nnn** is the 3 digit decimal code for the character. The character set tables in the Appendix show the 3 digit code for each character.

A good application using the Symbol or ZapfDingbat fonts would be in a RECODE statement to replace certain values with special characters or a combination of values and special characters. In the following report showing information about bank loans, we use a special arrow to point out loans that mature in December (month 12). In addition, for loans with missing maturity dates, we replace the values with a "telephone" character to remind us to call someone and find out why this information is missing.

Example

```
RECODE MAT_SYM 'Month Maturing' ON MAT_MO;
FONT D '\037'                          IF '00';
                                       IF '  ';
FONT D '\234' FONT RESET VALUE         IF 12;
VALUE                                  IF OTHER;


REPORT F2 'Report on the Status of Selected Loans':
        LOAN_NO THEN PL_CLASS THEN BAL THEN MAT_SYM;
```

Report on the Status of Selected Loans

Row	Loan Number	Classification	Balance	Month Maturing
1	4	Unknown	$1,000	☎
2	36	1 to 4 Family	$3,144	01
3	43	Unknown	$12,900	☎
4	46	Unknown	$14,000	☎
5	75	1 to 4 Family	$11,135	03
6	93	1 to 4 Family	$4,747	➪12
7	98	1 to 4 Family	$4,325	➪12
8	106	1 to 4 Family	$3,921	☎
9	133	1 to 4 Family	$4,925	➪12
10	142	1 to 4 Family	$47,770	01

Note that most Symbol and ZapfDingbats characters cannot be printed if you switch from PostScript to line printer mode. They will print as blanks or other characters, depending on your printer's character set.

Spaces in Proportional Fonts

With a proportional font, a blank space cannot be the same width as all other characters, because the character widths vary. In general, a blank is approximately one half the width of a number in the same font -- or one half the average width of a letter. Thus, with a proportional font, you will need about twice the number of blanks to get the same amount of blank space you would get with a non-proportional font.

Default Font defaults are initially set at installation time. If you install TPL REPORT for use with a PostScript printer, the following FORMAT statements will be included in your profile.

```
Postscript = yes;
Default font = H 10;
Footnote text font = H 8;          (currently unused by TPL REPORT)
Footnote symbol font = H 8;        (currently unused by TPL REPORT)
Title font = HB 12;
```

Note that if you have these statements in your profile, you must include FONT statements in your format requests to override any of the profile fonts that you wish to change. For example, if you change only the default font in your format request, you will still get the title font as specified in the profile.

If you usually use fonts that are different from those that were established at installation time, you will probably want to delete or replace the font statements in the profile.

If you install TPL REPORT for use with non-PostScript printers, you can change to PostScript mode by adding the statement

```
POSTSCRIPT = YES;
```

in your profile or format request. The default font will then be C 12 (Courier 12) for all reports unless you also add your own font choices.

Example Following is a sample format request with font specifications:

```
postscript = yes;
default font H 10;
title font TB 12;
page width = 8.5 inches;
page length = 11 inches;
right margin = .7 in;
left margin = .7 in;
```

Recommendation

Helvetica is a good font for the data part of a report, especially when a small font (below 10 point) is used. Other fonts can be used effectively for the titles and other labels. Bold or italics may be used to emphasize certain text or data values.

MARGINS (LEFT, RIGHT, TOP, BOTTOM)

Format There are four MARGIN actions.

```
LEFT MARGIN = amount  [ unit ];
RIGHT MARGIN = amount  [ unit ];
TOP MARGIN = amount  [ unit ];
BOTTOM MARGIN = amount  [ unit ];
```

where **amount** is a number and **unit** is optional. If no unit is specified, **characters** are assumed. If a unit is specified, the amount can be a decimal number and unit can be expressed as **inches**, **cm** or **points**.

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning Leave a margin of the size indicated by n. One or more of the margins can be changed for a report. The margins do not have to be the same size. The report is positioned (centered or aligned left or right) within the space remaining after the left and right margins sizes are subtracted from the page width. The report begins on the first line after the top margin and breaks at the bottom margin if it is longer than one page.

Level Margins can be controlled at the individual report level. Margins cannot change within a report.

Default

```
LEFT MARGIN = 5;
RIGHT MARGIN = 5;
TOP MARGIN = 6;
BOTTOM MARGIN = 6;
```

Example

```
TOP MARGIN = 2 cm;
BOTTOM MARGIN = 3 cm;
LEFT MARGIN = 0;
RIGHT MARGIN = 0;
```

Effect Set the top margin to 2 cm and the bottom margin to 3 cm. Remove the left and right margins by setting them to 0.

To remove ALL margins, set all margins to 0. Do not remove margins if you are working in PostScript mode.

Restrictions The page must be wide enough to hold the NUMBER column, if present, + the margins + the widest column of data.

The default margins will work correctly in PostScript mode. If you want to change the margins with the MARGIN statement, we recommend that you express margin sizes in terms of inches, cm or points so that their absolute sizes for printing will not depend on the font size in effect. If margin sizes are expressed in terms of characters, the results will sometimes be acceptable, but they will often be something other than what you intended and, at worst, you will get report output that looks "buggy". If parts of a report are "lost" at the top or right edges of the paper, check your margin specifications.

For most laser printers, a margin size of at least .25 inches is required. If you try to print something that fills the paper to the edges, you may lose part if it.

NUMBER (LEFT, RIGHT, BOTH)

Format

```
NUMBER LEFT;
NUMBER RIGHT;
NUMBER BOTH;
```

Meaning NUMBER refers to a report column that contains the row number for each row of data in a report. By default, this column is added to the **left** side of the report.

You can move the location of the NUMBER column to the **right** side of the report using the NUMBER RIGHT statement. You can also request that the NUMBER column be shown on **both** the left and the right by using the NUMBER BOTH statement.

For banked tables, the NUMBER column is placed in the same location for each bank.

If you want to delete the NUMBER column completely, use the statement:

```
FOR VARIABLE NUMBER: DELETE COLUMN;
```

Level The position of the NUMBER column can be specified at the individual report level.

Default

```
NUMBER LEFT;
```

Example

```
FOR REPORT 1:  NUMBER BOTH;
```

Effect The first report has the NUMBER column on both sides of the report with the default label "Row" for each side.

NUMBER column on both sides of the report.

Row	Region	Commodity	Dollars	Row
1	5 ASIA	041 WHEAT UNMILLED	6,519,575	1
2	5 ASIA	044 CORN OR MAIZE UNMILLED	448,407	2
3	2 MEXICO,CENTRAL AM. & CARIBBEAN	081 FEEDING-STUFF FOR ANIMALS	239,013	3
4	3 SOUTH AMERICA	081 FEEDING-STUFF FOR ANIMALS	2,037,000	4
5	4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	34,603,211	5
6	4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	3,415,548	6
7	4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	5,564,700	7
8	4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	8,991,990	8
9	5 ASIA	081 FEEDING-STUFF FOR ANIMALS	6,388,668	9

Example

```
FOR REPORT 2:  NUMBER RIGHT;
FOR REPORT 3:  NUMBER BOTH;
```

Effect The first report will have the NUMBER column in the default location at the left. The second report will have the NUMBER column on the right. The third report will have the NUMBER column on both sides of the report.

PAGE LENGTH

Format

```
PAGE LENGTH = amount  [ unit ];
```

where **amount** is a number and **unit** is optional. If no unit is specified, lines are assumed. If a unit is specified, the amount can be a decimal number and unit can be expressed as **inches**, **cm** or **points**.

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning The report is divided into pages according to the number of lines per page specified by **amount**. The number of lines available for the report is determined by subtracting the top and bottom margins from the page length. For the second and following pages of a report, the title and heading labels are repeated.

Level Page length is controlled at the request level. All reports within the same report request will use the same page length specification.

Default The system default is set at installation time and stored in the file called **profile.tpl**. When the system is installed, **profile.tpl** is stored in the TPL system directory.

You can change the system default by editing the PAGE LENGTH specification in **profile.tpl** and saving the result back in the system directory.

If you want to leave the system default as is but change the default for a set of report requests, you can do so by making a copy of **profile.tpl** with a different PAGE LENGTH specification and saving it in the directory where you are running your report jobs.

Example

```
PAGE LENGTH = 50;
TOP MARGIN = 3;
BOTTOM MARGIN = 2;
```

Effect Reports will be divided into pages of length 50. The top and bottom margins will use a total of 5 lines, leaving 45 lines per page for the reports.

Restrictions The page must be long enough to hold 1 row of data plus the margins, report title and heading labels.

If you are working in PostScript mode, you should express page length in something other than lines. This is because, with PostScript, you can choose different character sizes. If page length is expressed in lines, the length of the page will vary as the character size changes. This result is usually undesirable. Specify page length in inches, cm or points, or use the PAPER statement to select a page size.

PAGE LENGTH AUTOMATIC

Format

PAGE LENGTH = AUTOMATIC;

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether. **AUTO** can be used as an abbreviation for **AUTOMATIC**.

Meaning

For each report, the page length will be set at the length needed to print the entire report without page breaks. This statement should not be used with banked reports. If your report is wide, make sure that the page width is large enough to hold all columns without banking.

PAGE LENGTH AUTOMATIC; is most useful in the following two cases:

1. If you are customizing your report output for use with other software and want to get an unbroken stream of data rows.

2. If you are reviewing a report on the screen and want to look at it as one long page, uninterrupted by the extra space, title and heading labels that would otherwise appear at each page break. For this purpose, you may find it convenient to use **PAGE WIDTH = AUTOMATIC;** to prevent banking of a wide report. The combination of automatic page length and automatic page width will allow you to review reports on the screen without the reports being split into sections as would be required for printing on a particular size of paper.

Level

Page length is controlled at the request level. All reports within the same report request use the same page length specification.

Default

Reports break at the end of each page.

Example

PAGE LENGTH = AUTOMATIC;

Effect

Each report will be formatted as one long page.

Restrictions

This statement cannot be used in PostScript mode.

This statement should not be used with banked reports.

PAGE MARKER

Format

```
PAGE MARKER = marker specification;
BOTTOM PAGE MARKER = marker specification;
```

Normally, there can be only one PAGE MARKER for a report. For PostScript reports, if both PAGE MARKER and BOTTOM PAGE MARKER are used, there can be one marker at the top of the page and another at the bottom.

The **marker specification** can be one or more of the following:

```
NUMBER
COUNT
START = n
TOP
BOTTOM
RIGHT
LEFT
RIGHT THEN LEFT
LEFT THEN RIGHT
DATE
TIME
JOB
ODD
EVEN
label segments
```

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether. The specifications can be in any order.

Meaning

PAGE MARKER is used to add identifying information to report pages. The page marker can contain any combination of page number, date, time, job id and label segments. Starting page number can also be set.

Level

PAGE MARKER can be specified for each report separately. However, start number and marker location will carry forward to following reports unless they are explicitly reset. Further, the start number and marker location are independent of marker text even though they may appear in the same FORMAT statement.

Example

```
PAGE MARKER = TOP LEFT NUMBER START 3;
FOR REPORT 2: PAGE MARKER = START 5 "Page " NUMBER;
```

The result will be that all page markers will be in the top left corner of the page. Report 1 will just have numbers starting at 3. Report 2 will have "**Page** " and

numbers starting at 5. Report 3 and following reports will just have numbers, but the numbers will start where report 2 left off since there is no new START term for report 2.

Default Reports do not have page markers. If PAGE MARKER is specified, the default page start is **START = 1** and the default marker location is **TOP CENTER**. A BOTTOM PAGE MARKER is always at the bottom of the page with a default alignment of **CENTER**.

Page Numbering

PAGE MARKER = NUMBER; will produce page numbers that are centered vertically and horizontally within the top margin of a page. **PAGE MARKER = NUMBER START 5;** will cause the first report to start numbering at 5. Succeeding reports will continue the numbering unless a new **PAGE MARKER = START n;** is specified.

Example

```
FOR REPORT 1:  PAGE MARKER  RIGHT  'A'  NUMBER;
FOR REPORTS 2 AND 3:  PAGE MARKER  RIGHT  'B'  NUMBER  START 1;
```

Effect This example uses a combination of a letter and a page number to mark groups of report pages. Thus, for example, if the first report is to be inserted in Section A of a document, the page markers can be "A1", "A2", "A3", etc. If the second and third reports are to be inserted in Section B of the same document, they can have markers of "B1", "B2", "B3", etc. Since START 1 is included in the marker, the numbering for the second and third reports will restart at 1 on the first page of the second report; the numbering will continue on through the third report. All page markers will be in the top right corner of the page.

ODD and EVEN

When a page marker for a report includes both NUMBER and ODD, the page numbers for that report will all be odd. For example, if ODD is specified for the first report, its pages will be numbered 1,3,5,7,...

Use of EVEN will result in even numbered report pages.

If a report would normally begin with an even number but ODD is specified, then one is added to its starting number so that it will begin with an odd number.

Example

```
FOR REPORT 2:  PAGE MARKER = "PAGE " NUMBER START 7 EVEN;
```

Effect Page numbering for report 2 will begin with 8, the first even number after the specified start number.

ODD and EVEN are useful when a document has reports on every other page or when wide reports are displayed as facing page pairs. This latter is done in TPL REPORT by creating two reports with corresponding stubs and the heading split across the two reports. The first report should have a page marker which includes EVEN while the second should have the same starting page but should include ODD in its page marker.

Page Count

COUNT can be used to get a page count. It specifies the total number of report pages produced by a job. It is not affected by START or by the presence of multiple Page Markers in the job. An example of a statement using COUNT is:

Example

```
PAGE MARKER = "Page " NUMBER " of "  COUNT;
```

If there are 10 pages of report output in the job, the marker for the first page will be "Page 1 of 10"; the marker for the second page will be "Page 2 of 10"; and so on to the last page with a marker of "Page 10 of 10".

Marker Location

The location of page markers can be controlled by using TOP, BOTTOM, LEFT, RIGHT, or CENTER. CENTER is the default location. If LEFT is specified, the marker will start on the left page margin. If RIGHT is specified, the marker will end on the right margin. If TOP is specified, the marker will be placed 1/2 of the top margin down from top of page. If BOTTOM is specified, the marker will be placed 1/2 of the bottom margin above the bottom of the page.

If you are using TOP or BOTTOM, you may wish to increase the top or bottom margin specification beyond the standard 1 inch to keep the markers from appearing too near the top or bottom edge of the paper. This is especially important if you are using a multiline marker with a laser printer, since laser printers do not print on the top and bottom 1/4 inches of the paper.

If you want marker locations to alternate between left and right pages, use LEFT THEN RIGHT or RIGHT THEN LEFT.

As with START number, the marker location will continue across reports unless a new location is specified. For example, if you begin with RIGHT THEN LEFT for the first report, the marker location will alternate between right and left pages for all following reports unless an explicit LEFT, RIGHT, or CENTER is specified in a FORMAT statement for a later report.

Multiple Page Markers

If there are multiple PAGE MARKER statements for a report, each one will override the preceding one so that there will be only one marker. For PostScript reports, you can have two markers, one at the top of the report and one at the bottom, by using both a PAGE MARKER and a BOTTOM PAGE MARKER. The same options are available for both the top and bottom markers.

Example

```
PAGE MARKER RIGHT THEN LEFT "Research Bulletin ATN-05";
BOTTOM PAGE MARKER "Page " NUMBER;
```

Effect The text "Research Bulletin ATN-05" will be displayed at the top right of the first page, the top left of the second page, and so on. The first page will also have "Page 1" centered at the bottom, the second page "Page 2", and so on.

Notes If you specify BOTTOM in a regular page marker and also have a BOTTOM PAGE MARKER statement for the same report, both markers will go at the bottom with one possibly overlaying part of the other.

Alignments and Spacing within Page Markers

LEFT, RIGHT, or CENTER can only be used at the beginning of a marker, before any label segments. This alignment applies to the entire page marker. If you want more control of spacing within the marker, see SPACE and SPACE TO in the chapter called "Labels".

Other Options

In addition to or instead of page numbers, a page marker can contain anything allowed in a label except a footnote. Also there are some built-in special items. These are DATE, TIME, and JOB.

Example

```
PAGE MARKER = TOP LEFT THEN RIGHT
"Page " NUMBER " for job " JOB
" run on " DATE " at " TIME;
```

Effect If job TPLR873 is run on March 4, 2003 at 11:24 A.M., the output will be:

```
Page 1 for job TPLR873 run on 3/4/03 at 11:24:00 AM
```

which will appear in the upper left corner of Page 1. Page 2 will have its marker in the upper right corner of the page.

Note that the format of date and time will be affected by the COUNTRY statement. The examples in this section are shown in the format for the default country, **COUNTRY = US;**.

Since page markers are vertically centered within their margin, use of slashes at the start of a page marker specification will push the marker text down while slashes at the end of the page marker specification will raise the marker.

4-Digit Year

You can choose to have year displayed with 4 digits by editing the file called **country.tpl**. This file is installed in the TPL system directory, but you may also have customized copies in other directories where you run TPL jobs.

The **country.tpl** file is a simple ascii text file. The following options for date format are shown near the top of the file:

```
*  date format code
*    0 -> mm/dd/yy
*    1 -> dd/mm/yy
*    2 -> yy/mm/dd
*    3 -> mm/dd/yyyy
*    4 -> dd/mm/yyyy
*    5 -> yyyy/mm/dd
```

To choose 4-digit year as your standard, edit the entry for your country by changing the value in the fourth column to the number that matches the date format you want.

All current dates displayed in your TPL jobs will show the year in four digits. These include run dates in the output file as well as dates that are specified with the PAGE MARKER statement.

UNIX Users: Add a COUNTRY statement in your **profile.tpl** file if you do not already have one. For example:

```
COUNTRY = US;
```

Windows Users: If you make changes to **country.tpl**, add a COUNTRY statement to your profile, or change a COUNTRY statement in your profile, you need to ***restart TPL*** to activate the changes.

PAGE WIDTH

Format

```
PAGE WIDTH = amount  [ unit ];
```

where **amount** is a number and **unit** is optional. If no unit is specified, **characters** are assumed. If a unit is specified, the amount can be a decimal number and unit can be expressed as **inches**, **cm** or **points**.

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning

The report is formatted to fit within a page width of **n** characters. The report is aligned within the space remaining after the left and right margins are subtracted. If the report is too wide for the space, it is divided into as many partitions (called "banks") as necessary with each partition beginning on a new page. The NUMBER column, if present, is repeated in each partition.

Level

Page width is controlled at the request level. All reports within the same report request will use the same page width specification.

Default

The system default is set at installation time and stored in the file called **profile.tpl**. When the system is installed, **profile.tpl** is stored in the TPL system directory.

You can change the system default by editing the PAGE WIDTH specification in **profile.tpl** and saving the result back in the system directory.

If you want to leave the system default as is but change the default for a set of report requests, you can do so by making a copy of **profile.tpl** with a different PAGE WIDTH specification and saving it in the directory where you are running your report jobs.

Example

```
PAGE WIDTH = 100;
LEFT MARGIN = 2;
```

Effect

The report will be formatted within a page width of 100 characters. It will be centered within the 96 character space remaining after the left and right margins are subtracted from the page width.

Restrictions

If you are working in PostScript mode, you should express page width in something other than characters. This is because, with PostScript, you can choose different character sizes. If page width is expressed in characters, the width of the page will vary as the character size changes. This result is usually undesirable. Specify page width in inches, cm or points, or use the PAPER statement to select a page size.

PAGE WIDTH AUTOMATIC

Format PAGE WIDTH = AUTOMATIC;

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether. **AUTO** can be used as an abbreviation for **AUTOMATIC**.

Meaning TPL REPORT calculates the page width to be the sum of the widths of all columns, and the left and right margins.

Level Page width is controlled at the request level. All reports within the same report request will use the same page width specification.

Default The system default is set at installation time and stored in the file called **profile.tpl**. When the system is installed, **profile.tpl** is stored in the TPL system directory.

You can change the system default by editing the PAGE WIDTH specification in **profile.tpl** and saving the result back in the system directory.

If you want to leave the system default as is but change the default for a set of reports, you can do so by making a copy of **profile.tpl** with a different PAGE WIDTH specification and saving it in the directory where you are running your report jobs.

Example (for a report with 6 columns, including the NUMBER column):

```
PAGE WIDTH = AUTOMATIC;
LEFT MARGIN = 4;
RIGHT MARGIN = 4;
COLUMN WIDTH = 15;
```

Effect The report will be formatted for a page width of 98 characters (4 + 4 + 15*6).

PAPER

Format

PAPER = papersize;

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning

PAPER can be used to select one of the standard built-in paper sizes. Options are:

LETTER	(8.5 in x 11 in)
LEGAL	(8.5 in x 14 in)
A3	(42.0 cm x 29.7 cm)
A4	(21.0 cm x 29.7 cm)
B5	(18.2 cm x 25.7 cm)

You can choose one of these paper sizes at installation time or by adding the PAPER statement to your profile or format request.

The PAPER statement is used in place of the combination of PAGE WIDTH and PAGE LENGTH. For example, the statement

```
PAPER = LETTER;
```

gives the same result as the pair of statements

```
PAGE WIDTH = 8.5 IN;
PAGE LENGTH = 11 IN;
```

To determine the amount of space available for the report, deduct the margins from the page size. The reports will begin on the first line after the top margin and will be aligned within the left and right margins.

Level

Paper size is controlled at the request level. All reports within the same report request will be formatted for the same page size.

Default

The system default is set at installation time and stored in the file called **profile.tpl**. When the system is installed, **profile.tpl** is stored in the TPL system directory.

You can change the system default paper size after installation by editing **profile.tpl**.

Example

```
PAPER = LETTER;
```

Effect

All reports will be formatted for letter size paper (8 1/2 in x 11 in).

POSTSCRIPT

Format

POSTSCRIPT = YES; or

POSTSCRIPT = NO;

The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning

If you specify

POSTSCRIPT = YES;

report output will be coded in PostScript and all printing done from TPL REPORT will assume that you are working with a PostScript printer.

If you specify

POSTSCRIPT = NO;

your reports will be formatted in line printer mode and all printing done from TPL REPORT will assume that you are working with a line printer. FORMAT statements that apply only to PostScript will be ignored.

In PostScript mode, you can choose from any of the PostScript fonts available on your printer. See the FONT statement for details on font selection. Most of the fonts are proportional. This means that the character widths vary from character to character. For example, the letter "**i**" takes up less space than the letter "**m**". TPL REPORT will do all of the format adjustments needed for proper alignment with proportional fonts.

You can change the overall line spacing with the EXTRA LEADING statement. Line spacing for a particular line will also be affected by the fonts used for different parts of the line. For labels, spacing is determined by the largest font used in the label. For data rows, if the DEFAULT FONT is larger than any of the label fonts used for the row, the DEFAULT FONT will determine the line spacing. See FONT rules for labels and masks for additional details.

You can also print your reports sideways on the page. See the ROTATE statement for details.

In PostScript mode, all lines dividing sections of a report are drawn as solid lines.

For a complete list of FORMAT statements that are effective with POSTSCRIPT = YES; see the "PostScript" chapter.

Interaction of Size Specifications with PostScript

Following is a list of the size specifications that can be affected by a change to PostScript mode. If these sizes are expressed in terms of characters or lines, rather than in centimeters, inches or points, the absolute sizes for printing will depend on the font size in effect.

```
PAGE LENGTH = size;
PAGE WIDTH = size;
COLUMN WIDTH = size;
TOP MARGIN = size;
BOTTOM MARGIN = size;
LEFT MARGIN = size;
RIGHT MARGIN = size;
```

Size can be specified as a number followed by:

```
inch
inches
in
ins
cm
points
pt
pts
```

Fractional sizes must be specified as decimal numbers. For example,

```
COLUMN WIDTH = 2.5 IN;
```

In general, if you will be switching between line printer and PostScript mode, sizes other than page and margin size will work well in both modes if they are expressed in characters. If you are using a proportional font in PostScript, you will often be able to get more characters within a given width. The most common exception is when you have a label in upper case letters. Upper case letters are often wider in a proportional font.

Sizes specified in inches, centimeters or points will work in line printer mode as well as in PostScript mode. If you are not requesting PostScript output, the measures will be converted to 12 pt equivalents in characters. With 12 pt type, 1 inch can contain 10 characters in the horizontal direction and 6 lines in the vertical direction.

Page and Margin Sizes

In PostScript mode, page and margin sizes should be expressed in terms of centimeters, inches or points so that their absolute sizes for printing will not depend on the font size in effect. In the case of page size, you can also use the PAPER statement to pick a standard paper size.

The system default margins will work correctly in PostScript mode. If you want to change the margins with the MARGIN statement, we recommend that you express margin sizes in terms of inches, cm or points.

If your report is not positioned properly on the paper or if parts of the report are "lost" at the top or right edges of the paper, check your page and margin specifications.

Level PostScript is controlled at the request level. If PostScript is specified, all reports in the request will be created in PostScript mode.

Default The default is set at installation time.

If you are working with a PostScript printer and would like to have PostScript defaults entered in your system profile, you can set these defaults when you install TPL REPORT. See Installation Instructions for details. You can change any of these defaults after installation by editing **profile.tpl**, or you can override them with FORMAT statements in your format requests.

Example

```
POSTSCRIPT = YES;
FOR REPORT 2:  ROTATE;
```

Effect All reports will be prepared in PostScript format. The second report will be rotated to print sideways on the page. All other reports will be printed upright.

Restrictions Most laser printers require a margin. If you try to print something that fills the paper to the edges, you may lose part if it. Therefore, we do not recommend margin sizes of 0 when using PostScript.

The DATA REPORT and PAGE LENGTH AUTOMATIC statements cannot be used in PostScript mode.

The statement DELETE LAST RULE is ignored in PostScript mode.

PRINT (UNIX only)

Normally, TPL REPORT will prompt you at the end of a job to find out whether you want to print your **output** file or your reports. You can use the following statements to select the print options in advance.

Format

```
PRINT OUTPUT = YES or NO or PROMPT;
PRINT REPORTS = YES or NO or PROMPT;
```

Default PRINT statements are included in **profile.tpl** by the TPL installation process. The word REPORTS is a synonym for the word TABLES, but TABLES is the word used by the installation process. If you have installed TPL TABLES and TPL REPORT in the same subdirectory, the profile is shared by the two systems. The PRINT statements apply equally to TPL TABLES and TPL REPORT.

The default for both statements is **PROMPT**.

PRINT COMMAND (UNIX profile only)

Format PRINT COMMAND = ‘command‘ ;

where **command** is a UNIX print command.

Meaning TPL REPORT will direct its output to the default printer for your computer. If you wish to change this, you may modify the PRINT COMMAND statement in the profile.

Level The command takes effect for the entire report request.

Default PRINT COMMAND = ‘lp’;

Example PRINT COMMAND = ‘lp -dpost’;

where **post** is the name of your PostScript printer.

Effect Reports and other output will be directed to the PostScript printer.

Note Unlike most FORMAT statements, PRINT COMMAND will only work if it is placed in the profile, not in the FORMAT request. If different people wish to use different printers, they should create local profiles with different print statements.

REPLACE COLOR

REPLACE COLOR is useful when pre-viewing color tables on a monochrome printer, because it lets you replace colors with special fonts. See the FORMAT statement called COLOR = NO for details.

REPLACE DIVIDE CHARACTER

Note This statement is ignored in PostScript mode.

Format

```
REPLACE DIVIDE CHARACTER WITH 'char';
```

Meaning Replace the column dividers and the side rules, if present, with the character enclosed in quotes.

The statement takes effect only if down rules are retained and you are operating in line printer mode. Down rules can be retained with RETAIN DOWN RULES or RETAIN ALL RULES.

Level The divide character can be controlled at the individual report level. The divide character cannot change within a report.

Default

```
REPLACE DIVIDE CHARACTER WITH '|';
```

Example

```
RETAIN DOWN RULES;
REPLACE DIVIDE CHARACTER WITH '*';
```

Effect The column dividers will be replaced with vertical lines of the character *****.

Restrictions The divider can be only one character wide and cannot be a null character ''.

REPLACE LABEL

Format FOR VARIABLE variable name: REPLACE LABEL WITH label;

Meaning The label will be replaced for all occurrences of the variable named in the FOR clause. Report names or numbers may also be referenced in the FOR clause to restrict the label replacement to one or more specific reports. If no reports are mentioned, the changes occur for all reports in which the named variables are used.

Level Variable labels can be controlled at the report level. A particular variable cannot have more than one label within a report.

Default The default variable label is determined when the variable is described in the codebook or defined in the report request.

Example

```
FOR REPORT 1 VARIABLE Employees:
        REPLACE LABEL WITH 'Workers';
FOR REPORT 2 VARIABLE NUMBER:
        REPLACE LABEL WITH 'Report Row Number';
```

Effect In the first report, the label for the variable **Employees** will be replaced with the label **Workers**. In the second report, the label for the built-in variable **NUMBER** will be replaced with the label **Report Row Number**. If the NUMBER column appears more than once in the report, the replacement label will be used on all NUMBER columns.

Note In report cells, values can be replaced by value labels from the codebook or by entirely new labels with the RECODE statement. See the chapter called **RECODE** for details.

REPLACE MASK

Format REPLACE MASK WITH mask;

Meaning Replace the mask for *observation* variables with a new one. The new mask can be any valid TPL REPORT mask. If no FOR clause is used, the mask will apply to all observation variables in all reports, including the built-in NUMBER variable. You can restrict the application of the mask either by location OR by variable but not both in the same FORMAT statement.

Alignments specified in masks interact with ALIGN statements such as ALIGN COLUMN. If both a mask and an ALIGN specification apply to the same column, the alignment in the last specification wins.

Default The default mask is established when an observation variable is described in the codebook or computed in a report request. If no mask is associated with an observation variable, its cell values are displayed centered and rounded to the nearest whole integer with no other special symbols except commas.

Replacing Mask by Location

To replace masks for particular report columns, use a FOR clause with the appropriate column location.

Level The location for mask replacement can be specified at the individual column level.

Example
```
FOR COLUMN 2:  REPLACE MASK WITH '$'9,999.99;
```

Effect The mask will be replaced to show dollars and cents in the second column of all reports.

Example
```
FOR REPORTS 2 AND 3 COLUMNS 3 TO 6:
REPLACE MASK WITH 999.99 RIGHT;
```

Effect The values in columns 3 through 6 of reports 2 and 3 will be right-adjusted in the columns and displayed to show two decimal places.

Example
```
FOR REPORT B1 COLUMN 2:  REPLACE MASK WITH '$'999,999 RIGHT;
FOR REPORT B1 COLUMN 1:  REPLACE MASK WITH 'Secret';
```

Effect The values in column 2 of report B1 will be **right**-adjusted and displayed with a **$**. The values in column 1 of report B1 will be replaced by the word **Secret**.

Before

Report on bank loans showing current balances.

Row	Original Loan	Loan Balance	Loan ID
1	4,000	1,000	00004
2	16,300	5,743	00008
3	14,000	1,546	00011
4	7,600	3,144	00036
5	16,200	3,065	00040
6	13,000	2,594	00041
7	14,800	3,009	00043
8	27,900	12,900	00043
9	100,000	100,000	00044
10	150,000	15,000	00045

After

Report on bank loans showing current balances.

Row	Original Loan	Loan Balance	Loan ID
1	Secret	$1,000	00004
2	Secret	$5,743	00008
3	Secret	$1,546	00011
4	Secret	$3,144	00036
5	Secret	$3,065	00040
6	Secret	$2,594	00041
7	Secret	$3,009	00043
8	Secret	$12,900	00043
9	Secret	$100,000	00044
10	Secret	$15,000	00045

Replacing Mask by Variable

To replace the mask for an observation variable, use a FOR clause with the variable name.

Level Variable masks can be replaced for individual reports.

Example

```
FOR VARIABLE INCOME:  REPLACE MASK WITH '$'999,999.99;
```

Effect The mask for the observation variable INCOME will be replaced in any reports where INCOME is used.

Example

```
FOR REPORTS 2 AND 3, VARIABLE INCOME:
        REPLACE MASK WITH 9,999.99 RIGHT;
```

Effect The INCOME values in reports 2 and 3 will be right-adjusted in the columns and displayed to show two decimal places. If INCOME is used in any other reports, the mask will not be replaced for those reports.

Treatment of Conflicting Masks

Mask replacement cannot be specified both by variable and by column location:

1. If the two types of specification are used in the **same** FOR clause, any location specification other than report will be ignored and the mask will be replaced wherever the variable is used.

2. If the same report location would be affected by two **different** REPLACE MASK statements, where one is specified by variable and the other is specified by column, the last statement wins.

Restrictions Masks cannot be replaced for:

1. Control, CHAR or RECODE variables; (Note however that alignments can be changed for these variables. See ALIGN CELL and ALIGN COLUMN.)

2. any report cells with special indicators such as (c) for computation error;

3. report cells specified by row rather than column, since rows cannot be referenced in report FORMAT statements.

REPLACE MASK COLOR

Format

```
REPLACE MASK COLOR WITH  color-name;
REPLACE MASK COLOR WITH  r g b;
```

where

r, **g** and **b** are numbers between 0 and 100 (inclusive) which specify red, green, and blue components of color;

color-name is the name of a color defined in the **color.tpl** file.

The word CELL is a synonym for the word MASK.

Meaning This statement lets you replace the color of a mask without disturbing any other specifications in the mask and without re-entering the entire mask. *Unlike most MASK statements, this one applies to all types of variables, observation, control and char.*

Level Mask color can be replaced by column or for individual observation variables.

Example

```
FOR COLUMN 1:  REPLACE MASK COLOR WITH RED;
FOR VARIABLE INCOME:  REPLACE MASK COLOR WITH GREEN;
```

Effect The mask color for the first column will be red. The columns containing INCOME values will have a mask color of green.

Restrictions This statement does not affect the NUMBER column unless you use a statement that specifically names the NUMBER column. For example:

```
FOR VARIABLE NUMBER:  REPLACE MASK COLOR WITH RED;
```

Special cell indicators such as (f), for values that do not fit within the column width, are not affected by the statement. Colors for these indicators are controlled by SYMBOL COLOR as described for the COLOR Defaults statements.

If you have a REPLACE MASK statement following the REPLACE MASK COLOR statement, it will override the REPLACE MASK COLOR statement.

REPLACE MASK FONT

Format REPLACE MASK FONT WITH font-name [n];

where

font-name is a font identifier such as H or TB and

n is a number indicating a font size.

The word CELL is a synonym for the word MASK.

Meaning This statement lets you replace the font of a mask without disturbing any other specifications in the mask and without re-entering the entire mask. It applies only to *observation* variables. If you replace the mask font for all columns with observation values, you get the effect of changing the DEFAULT FONT for these columns without affecting the DEFAULT FONT as applied to any other parts of the reports.

Level Fonts can be replaced at the level of individual columns or for observation variables.

Example

```
FOR COLUMN 1:  REPLACE MASK FONT WITH B 12;
FOR VARIABLE INCOME:  REPLACE MASK FONT WITH B;
```

Effect The font for the first column will be Bookman 12. The columns containing INCOME values will have a mask font of Bookman, and the size will be whatever font size is already specified for these cells.

Example

```
DEFAULT FONT = T 12;
REPLACE MASK FONT WITH H 11;
```

Effect The font Times 12 will be used for all parts of the reports except the columns containing values for observation variables. The MASK FONT Helvetica 11 will be used for these columns.

REPLACE TITLE

Format REPLACE TITLE WITH label;

Meaning Replace the report title established in the REPORT statement with a new title. The title can be any valid TPL REPORT label.

Level The report title can be controlled at the report level.

Default The default report title is the one specified in the REPORT statement. If no report title is specified in the REPORT statement, the report name is used as the default report title.

Example REPLACE TITLE WITH CENTER 'Student Records.';

Effect The report title is replaced by a centered title

Student Records

REPLACE TITLE CONTINUATION

Format

```
REPLACE TITLE CONTINUATION WITH label;
```

CONTINUE and CONTINUED are synonyms for CONTINUATION.

Meaning Replace the default continuation indicator for the report title with a label. It will be added to the title on all pages after the first page of the report. The continuation indicator can be any valid TPL REPORT label.

If the title contains the keyword CONTINUATION, the continuation indicator will be inserted at that point. Otherwise, it will be added at the end of the title.

Level Title continuation can be controlled at the report level.

Default The default continuation indicator is ' **- Continued**'.

Example

```
REPLACE TITLE CONTINUATION WITH ' (Cont.)';
```

Effect The continuation indicator ' **(Cont.)**' will be added to the report title on pages following the first page. If the report title is:

```
Employee Records for Pay Period 12.
```

then the result on the second and following pages will be:

```
Employee Records for Pay Period 12. (Cont.)
```

Example

```
REPLACE TITLE CONTINUATION WITH '';
```

Effect The title continuation indicator will be suppressed.

RETAIN

The following statements are defaults. They are described with their corresponding DELETE statements. See the DELETE statements for details.

```
RETAIN COLUMNS;
RETAIN HEADING;
RETAIN LEADING ZEROS;
RETAIN REPORTS;
RETAIN TITLE;
```

RETAIN ALL RULES

Format RETAIN ALL RULES;

Meaning This statement is equivalent to the combination:

RETAIN DOWN RULES;
RETAIN CROSS RULES;

Horizontal lines called cross rules are displayed above and below the column head and across the bottom of the report, and vertical lines called down rules are displayed between columns.

Note This statement has no effect on side rules. To retain side rules as well and effectively "box in" the report except for the title, see the statement RETAIN SIDE RULES.

Level Rules can be retained for individual reports.

Default DELETE ALL RULES;

No rules are displayed unless side rules have been retained by a separate statement.

Example FOR REPORT 2: RETAIN ALL RULES;

Effect For the second report, cross rules and down rules will be displayed.

Shipping report.

Row	Commodity	Short Tons	Dollars
1	08 Animal feeds	214,295	34,603,211
2	22 Oil seeds, nuts & kernals	28,985	7,342,841
3	22 Oil seeds, nuts & kernals	63,729	12,122,062
4	28 Metalliferous ores & scrap	19,301	1,358,377
5	32 Coal, cokes, & briquettes	63,176	1,926,413
6	32 Coal, cokes, & briquettes	32,211	616,551
7	32 Coal, cokes, & briquettes	78,825	3,575,442
8	51 Organic chemicals	1,637	1,042,057

Example

```
RETAIN ALL RULES;
RETAIN SIDE RULES;
```

Effect All reports will be displayed with cross rules, down rules and side rules. The side rules will join with the top and bottom cross rules, so that they form a box around the report. The report title will be outside and above the box.

Shipping report.

Row	Commodity	Short Tons	Dollars
1	08 Animal feeds	214,295	34,603,211
2	22 Oil seeds, nuts & kernals	28,985	7,342,841
3	22 Oil seeds, nuts & kernals	63,729	12,122,062
4	28 Metalliferous ores & scrap	19,301	1,358,377
5	32 Coal, cokes, & briquettes	63,176	1,926,413
6	32 Coal, cokes, & briquettes	32,211	616,551
7	32 Coal, cokes, & briquettes	78,825	3,575,442
8	51 Organic chemicals	1,637	1,042,057

RETAIN BLANKS

Format RETAIN BLANKS;

Meaning This statement applies only to **CHAR** variables.

When CHAR variables are used in reports formatted for printing, leading and trailing blanks are removed from the values. In DATA REPORTS, the blanks are retained. You can use RETAIN (or DELETE) blanks to change the default treatment.

Level Treatment of blanks can be controlled at the column level.

Default DELETE BLANKS; if formatted for printing;
RETAIN BLANKS; if formatted for a DATA REPORT.

Example FOR REPORT 2, COLUMN 3: RETAIN BLANKS;

Effect If the third column in report 2 is a CHAR variable that has leading or trailing blanks in the values, the blanks will be retained.

Example FOR REPORT 2, VARIABLE COMPANY_NAME: RETAIN BLANKS;

Effect If COMPANY_NAME is a CHAR variable that has leading or trailing blanks in the values, the blanks will be retained.

RETAIN CROSS RULES

Format RETAIN CROSS RULES [WEIGHT = n] [DOUBLE or SINGLE] ;

where **n** is a number. The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning Horizontal lines called cross rules will be displayed above and below the column head and across the bottom of the report.

To retain other types of rules in a report, see also, RETAIN DOWN RULES, RETAIN SIDE RULES and RETAIN ALL RULES.

Note that cross rules are also retained with the statement RETAIN ALL RULES.

The optional WEIGHT specification can be used to control the thickness of the cross rules in PostScript mode. The weight is expressed in points where each point is 1/72 inches.

If DOUBLE is specified, the cross rules will be double lines.

See also, the statements RULE WEIGHT, DOWN RULE WEIGHT and RETAIN SIDE RULES for additional ways to control the thickness of rules in reports.

Level Cross rules can be retained for individual reports.

Default DELETE CROSS RULES;

No cross rules are displayed.

The default weight for cross rules is .5 unless the weight for all rules has been changed with the RULE WEIGHT statement.

Example FOR REPORT 2: RETAIN CROSS RULES;

Effect For the second report, cross rules will be displayed. No cross rules will be displayed in other reports.

Shipping report.

Row	Commodity	Short Tons	Dollars
1	08 Animal feeds	214,295	34,603,211
2	22 Oil seeds, nuts & kernals	28,985	7,342,841
3	22 Oil seeds, nuts & kernals	63,729	12,122,062
4	28 Metalliferous ores & scrap	19,301	1,358,377
5	32 Coal, cokes, & briquettes	63,176	1,926,413
6	32 Coal, cokes, & briquettes	32,211	616,551
7	32 Coal, cokes, & briquettes	78,825	3,575,442
8	51 Organic chemicals	1,637	1,042,057

Example
```
RULE EVERY 3;
RETAIN CROSS RULES WEIGHT = 1.5;
```

Effect A horizontal rule will be inserted every 3 rows. In addition, cross rules will be displayed above and below the column head and at the bottom of the report. These cross rules will be emphasized, because the weight of 1.5 is greater than the default of .5 that will apply to the rules between rows.

Regional shipping report.

Region	Commodity	Dollars
5 ASIA	041 WHEAT UNMILLED	6,519,575
5 ASIA	044 CORN OR MAIZE UNMILLED	448,407
2 MEXICO,CENTRAL AM. & CARIBBEAN	081 FEEDING-STUFF FOR ANIMALS	239,013
3 SOUTH AMERICA	081 FEEDING-STUFF FOR ANIMALS	2,037,000
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	34,603,211
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	3,415,548
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	5,564,700
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	8,991,990
5 ASIA	081 FEEDING-STUFF FOR ANIMALS	6,388,668

Restrictions The weight value must be greater than or equal to zero. A weight of 0 does not make the rules disappear. Instead, it results in the thinnest rules possible on your PostScript output device.

If the weight value is too large, the rules will be so thick that they will make broad bands that overlay the rows. This is usually undesirable. For example, the statement:

```
DOWN RULE WEIGHT = 72;
```

will create broad black bands that are 1 inch wide (1 pt. = 1/72", so 72 points = 1").

RETAIN DOWN RULES

Format RETAIN DOWN RULES;

Meaning The report columns will be separated by vertical rules (lines) from top to bottom. One space of each column is reserved for column separation.

To retain other types of rules in a report, see also, RETAIN CROSS RULES, RETAIN SIDE RULES and RETAIN ALL RULES.

Note that down rules are also retained with the statement RETAIN ALL RULES.

Level RETAIN DOWN RULES; can be restricted to specific columns, so you can retain them selectively. In the FOR clause, columns can be selected by column number or by variable name. When a column is selected, the rule following the column is retained. If the last column (right-most on the page) is selected, a down rule will not be retained for that column, since it is considered to be a "side rule". To retain side rules, see the statement RETAIN SIDE RULES.

Note If the **NUMBER** column is the left-most column (the default position for it), it does not have a column number. This means that column numbering starts at the first column following the NUMBER column on the left. In this case, the NUMBER column can only be referenced by name: FOR VARIABLE NUMBER.

Default DELETE DOWN RULES;

No down rules are displayed. The columns are separated by a blank space.

Example FOR REPORT 3: RETAIN DOWN RULES;

Effect For the third report, all columns will be separated by a vertical rule. For other reports, the columns will be separated by a blank space.

Shipping report.

Row	Commodity	Short Tons	Dollars
1	08 Animal feeds	214,295	34,603,211
2	22 Oil seeds, nuts & kernals	28,985	7,342,841
3	22 Oil seeds, nuts & kernals	63,729	12,122,062
4	28 Metalliferous ores & scrap	19,301	1,358,377
5	32 Coal, cokes, & briquettes	63,176	1,926,413
6	32 Coal, cokes, & briquettes	32,211	616,551
7	32 Coal, cokes, & briquettes	78,825	3,575,442
8	51 Organic chemicals	1,637	1,042,057

Example

```
FOR VARIABLE NUMBER AND COLUMN 1:
    RETAIN DOWN RULES;
```

Effect In all reports, down rules will follow the NUMBER column and column 1. Other columns will be separated by a blank character. Note that if the NUMBER column is the left-most column in a report (this is the default), it is considered to be column 0 but cannot be referenced as column 0. As shown in this example, you must reference the NUMBER column as VARIABLE NUMBER.

Shipping report.

Row	Commodity	Short Tons	Dollars
1	08 Animal feeds	214,295	34,603,211
2	22 Oil seeds, nuts & kernals	28,985	7,342,841
3	22 Oil seeds, nuts & kernals	63,729	12,122,062
4	28 Metalliferous ores & scrap	19,301	1,358,377
5	32 Coal, cokes, & briquettes	63,176	1,926,413
6	32 Coal, cokes, & briquettes	32,211	616,551
7	32 Coal, cokes, & briquettes	78,825	3,575,442
8	51 Organic chemicals	1,637	1,042,057

RETAIN SIDE RULES

Format RETAIN SIDE RULES [WEIGHT = n] [DOUBLE or SINGLE] ;

where **n** is a number. The word **IS** can be used in place of **=**. Both are optional and can be left out altogether.

Meaning This statement causes vertical rules (lines) to appear at the left and right edges of each bank of a report. Normally, one space of each column is reserved for column separation. When RETAIN SIDE RULES; is specified, one additional space must be reserved from each bank. This space is taken from the NUMBER column if it is present. Otherwise, it is taken from the first column of each bank.

To retain other types of rules in a report, see also, RETAIN DOWN RULES, RETAIN CROSS RULES and RETAIN ALL RULES.

Note that the statement RETAIN ALL RULES; has no effect on side rules.

The optional WEIGHT specification can be used to control the thickness of the side rules in PostScript mode. The weight is expressed in points where each point is 1/72 inches.

See also, the statements RULE WEIGHT, DOWN RULE WEIGHT and RETAIN CROSS RULES for additional ways to control the thickness of rules in reports.

Level Side rules can be retained for individual reports.

Default DELETE SIDE RULES;

No side rules are displayed.

The default weight for side rules is .5 unless the weight for all rules has been changed with the RULE WEIGHT statement.

Example FOR REPORT 2: RETAIN SIDE RULES;

Effect For the second report, vertical rules will be displayed on both sides of the report.

Shipping report.

Row	Commodity	Short Tons	Dollars
1	08 Animal feeds	214,295	34,603,211
2	22 Oil seeds, nuts & kernals	28,985	7,342,841
3	22 Oil seeds, nuts & kernals	63,729	12,122,062
4	28 Metalliferous ores & scrap	19,301	1,358,377
5	32 Coal, cokes, & briquettes	63,176	1,926,413
6	32 Coal, cokes, & briquettes	32,211	616,551
7	32 Coal, cokes, & briquettes	78,825	3,575,442
8	51 Organic chemicals	1,637	1,042,057

Example

```
RETAIN ALL RULES;
RETAIN SIDE RULES WEIGHT = 2;
```

Effect For all reports, cross rules, down rules and side rules will be retained. The side rules will be thicker than the other rules.

Restrictions The weight value must be greater than or equal to zero. A weight of 0 does not make the rules disappear. Instead, it results in the thinnest rules possible on your PostScript output device.

If the weight value is too large, the rules will be so thick that they will make broad bands that overlay the columns. This is usually undesirable. For example, the statement:

```
DOWN RULE WEIGHT = 72;
```

will create broad black bands that are 1 inch wide (1 pt. = 1/72", so 72 points = 1").

ROTATE

Note ROTATE is only effective in PostScript mode. It is ignored otherwise.

Format ROTATE;

Meaning The reports are formatted to print sideways on the page. This format is sometimes called "Landscape". Report margins are rotated along with the reports, so that the terms, **top**, **bottom**, **left** and **right**, are relative to the orientation of the report rather than the page.

If you have a report with many columns, you may be able to get all the columns on one page by using the ROTATE statement to turn the report sideways.

Level Rotation can be specified at the individual report level.

Default Reports are printed upright.

Example FOR REPORT 2: ROTATE;

Effect Report 2 will be rotated to print sideways. All other reports will print upright.

ONE

Row	COUNTRY	Dollars	Short Tons
1	542	6,519,575	54,710
2	588	448,407	3,662
3	225	239,013	1,143
4	307	2,037,000	9,700
5	421	34,603,211	214,295
6	428	3,415,548	22,214
7	470	5.564.700	28.495

TWO

Row	COUNTRY	Dollars	Short Tons
1	542	6,519,575	54,710
2	588	448,407	3,662
3	225	239,013	1,143
4	307	2,037,000	9,700
5	421	34,603,211	214,295
6	428	3,415,548	22,214
7	470	5,564,700	28,495

RULE EVERY

Format

RULE EVERY n;
or
RULE EACH n;

where **n** is a number. EVERY and EACH are synonyms.

Meaning

Insert a horizontal rule (line) every **n** rows. A row is defined as a data row in the report. If a row has any entries that take more than one line, it still counts as one row, so the rule will follow the bottom line for the row.

The rules are applied on a page by page basis. In other words, the row count starts over at the top of each page.

Level

RULE EVERY n; can be specified for individual reports.

Default

Reports are formatted without extra rules.

Example

FOR REPORT ONE: RULE EACH 1;

Effect

For Report One, horizontal rules are inserted between data rows.

Report One: Rule every row.

Region	Commodity	Dollars
5 ASIA	041 WHEAT UNMILLED	6,519,575
5 ASIA	044 CORN OR MAIZE UNMILLED	448,407
2 MEXICO,CENTRAL AM. & CARIBBEAN	081 FEEDING-STUFF FOR ANIMALS	239,013
3 SOUTH AMERICA	081 FEEDING-STUFF FOR ANIMALS	2,037,000
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	34,603,211
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	3,415,548
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	5,564,700
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	8,991,990
5 ASIA	081 FEEDING-STUFF FOR ANIMALS	6,388,668

Example

FOR REPORT TWO: RULE EVERY 3;
RETAIN CROSS RULES;

Effect For Report Two, horizontal rules are inserted every three data rows. Cross rules are also retained, so horizontal rules are included above and below the column headings and at the bottom of the report.

Report Two: Rule every third row.

Region	Commodity	Dollars
5 ASIA	041 WHEAT UNMILLED	6,519,575
5 ASIA	044 CORN OR MAIZE UNMILLED	448,407
2 MEXICO,CENTRAL AM. & CARIBBEAN	081 FEEDING-STUFF FOR ANIMALS	239,013
3 SOUTH AMERICA	081 FEEDING-STUFF FOR ANIMALS	2,037,000
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	34,603,211
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	3,415,548
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	5,564,700
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	8,991,990
5 ASIA	081 FEEDING-STUFF FOR ANIMALS	6,388,668

RULE WEIGHT

Format

RULE WEIGHT = n;

where **n** is a number. The word **IS** can be used in place of =. Both are optional and can be left out altogether. The word LINE can be used in place of the word RULE.

Meaning

This statement applies in PostScript mode only and only if you have used other FORMAT statements to put rules (lines) in your reports. RULE WEIGHT lets you adjust the thickness of the rules. The rule weight is expressed in points where each point is 1/72 inches.

The thickness of the rules will increase or decrease according to the rule weight number specified. Note that the appearance for a particular rule weight on the printed page will vary from printer to printer. This is especially true with printers of different dpi (dots per inch).

See also DOWN RULE WEIGHT and the WEIGHT options of RETAIN CROSS RULES and RETAIN SIDE RULES for adjusting the thickness of specific types of rules.

Level

Rule weight can be specified for individual reports.

Default

RULE WEIGHT = .5;

Example

RULE WEIGHT = .4;

Effect

All rules in the reports will have the same weight and will be slightly thinner than the default rule weight of .5;

Restrictions

The rule weight value must be greater than or equal to zero. A rule weight of 0 does not make the rule disappear. Instead, it results in the thinnest rule that is possible on your PostScript output device.

If the rule weight value is too large, the rules will be so thick that they will make broad bands that overlay other parts of the reports. This is usually undesirable. For example, the statement:

RULE WEIGHT = 72;

will create broad black bands that are 1 inch wide (1 pt. = 1/72", so 72 points = 1").

SKIP AFTER BANKS

Format SKIP n LINES AFTER BANK;

where **n** is a number.

Meaning If a report is too wide to fit on a page, it is automatically broken into sections called banks. Banking can also be requested explicitly with the **BANK AFTER COLUMN;** statement. By default, each bank begins on a new page. SKIP AFTER BANKS can be used to request a different spacing between banks so that more than one bank can be printed on a page.

If SKIP AFTER BANKS is specified by itself, TPL REPORT assumes that there should be two banks per page. The related statement **BANKS PER PAGE** can be used to request more than two banks per page.

All banks of a report are aligned the same way, but each bank is aligned independently. This means that if the default report alignment of CENTER is in effect, each bank will be centered on the page without regard to the placement of any other bank on the page. If, for example, you have specified ALIGN REPORTS LEFT, each bank will be aligned at the left margin of the page.

The report title will appear only once on a page regardless of the number of banks. It will be aligned relative to the first bank on the page and formatted according to the width of the first bank.

Note All banks on a given page will take up the same amount of vertical space. If a long data value requires more than one line, the corresponding row in other banks will take the same number of lines with blank lines inserted where necessary to keep the rows even between banks. In addition, if one bank has a very long heading label that must be broken into several lines, the space requirement for that label will apply to all of the banks. Each bank may then take more vertical space than you expect.

Level SKIP n LINES AFTER BANK can be specified for individual reports.

Default Each bank begins on a new page. If the BANKS PER PAGE statement is used without SKIP AFTER BANKS, the default is **SKIP 1 LINE AFTER BANKS;**

Example

```
BANKS PER PAGE = 3;
SKIP 2 LINES AFTER BANK;
```

Effect Each page of the report will contain 3 banks with 2 blank lines between the banks. If the number of report banks is not a multiple of 3, then the last page will contain fewer than 3 banks.

Restrictions There must be enough vertical space on the page for each bank to contain at least one row of data.

The number of lines between banks must be greater than 0.

SKIP LINE EVERY

Format

SKIP LINE EVERY n;
or
SKIP LINE EACH n;

where **n** is a number. **EVERY** and **EACH** are synonyms.

Meaning

Insert a blank line every **n** rows. A **row** is defined as a data row in the report. If a row has any entries that that take more than one line, it still counts as one row, so the blank line will follow the bottom line for the row.

The blank lines are applied on a page by page basis. In other words, the row count starts over at the top of each page.

Level

SKIP EVERY n; can be specified for individual reports.

Default

Reports are formatted without blank lines.

Example

FOR REPORT ONE: SKIP LINE EACH 1;

Effect

For Report One, blank lines are inserted between data rows.

Report One: Skip line every row.

Region	Commodity	Dollars
5 ASIA	041 WHEAT UNMILLED	6,519,575
5 ASIA	044 CORN OR MAIZE UNMILLED	448,407
2 MEXICO,CENTRAL AM. & CARIBBEAN	081 FEEDING-STUFF FOR ANIMALS	239,013
3 SOUTH AMERICA	081 FEEDING-STUFF FOR ANIMALS	2,037,000
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	34,603,211
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	3,415,548
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	5,564,700
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	8,991,990
5 ASIA	081 FEEDING-STUFF FOR ANIMALS	6,388,668

Example

FOR REPORT TWO: SKIP LINE EVERY 3;

```
RETAIN CROSS RULES;
```

Effect For Report Two, blank lines are inserted every three data rows. Cross rules are also retained, so horizontal rules are included above and below the column headings and at the bottom of the report.

Report Two: Skip line every third row.

Region	Commodity	Dollars
5 ASIA	041 WHEAT UNMILLED	6,519,575
5 ASIA	044 CORN OR MAIZE UNMILLED	448,407
2 MEXICO,CENTRAL AM. & CARIBBEAN	081 FEEDING-STUFF FOR ANIMALS	239,013
3 SOUTH AMERICA	081 FEEDING-STUFF FOR ANIMALS	2,037,000
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	34,603,211
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	3,415,548
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	5,564,700
4 EUROPE	081 FEEDING-STUFF FOR ANIMALS	8,991,990
5 ASIA	081 FEEDING-STUFF FOR ANIMALS	6,388,668

USE CONDITION LABEL, NAME, VALUE

Format

```
USE CONDITION LABELS;
USE CONDITION NAMES;
USE CONDITION VALUES;

REPLACE LABEL WITH 'label';
```

You can use either singular or plural words for LABELS, NAMES and VALUES. The word CONDITION is optional.

Meaning

For control variables, you can specify USE CONDITION to replace the values in a report with the condition labels or names from the codebook. The results are similar to what you would get in the report request if you used a RECODE statement to create a new variable with the condition labels or names of the old variable as values for the new variable. See the "Recode" chapters for details.

You must use a FOR clause that references one or more variables. Otherwise, you will get an error message. The statement is ignored for variables that are not control variables.

USE CONDITION NAMES is only effective if there are condition names in the codebook. USE CONDITION LABELS will choose the condition labels if present. If no labels are present, the condition names will be used. If no labels or names are present, a generated label will be created from the value.

The related statement, FOR CONDITION VARIABLE(n): **REPLACE LABEL WITH 'label';** can be used in conjunction with USE CONDITION LABELS to override specific values with labels of your choice. If the REPLACE LABEL statement is used alone or without a FOR clause to specify a condition, it will be ignored. If you have many values to replace, you may find it more convenient to use a RECODE statement.

Level

This statement can be specified for individual control variables at the report level.

Default

```
USE CONDITION VALUES;
```

Reports are formatted using condition values.

Example

```
FOR REPORT 1  VARIABLE SEX:  USE CONDITION LABELS;
```

Effect For the first report, the values in the SEX column will be replaced with the condition labels for SEX in the codebook.

Example

```
FOR REPORT 1  VARIABLE STATE:  USE CONDITION LABELS;
FOR REPORT 1 CONDITION STATE(1):  REPLACE LABEL WITH 'Montana';
FOR REPORT 1 CONDITION STATE(6):  REPLACE LABEL WITH 'Alabama';
```

Effect For the first report, the values in the STATE column will be replaced with the condition labels for STATE in the codebook. For all cells in the STATE column that have either the 1st condition value or the 6th condition value, the cell contents will be replaced by the labels 'Montana' and 'Alabama' respectively.

USE VARIABLE NAME

Format

```
USE VARIABLE NAME;
USE VARIABLE LABEL;
```

You can use the plural words NAMES and LABELS in place of NAME and LABEL.

Meaning If you have labels associated with variables that are used in a report, the labels will be printed at the top of the column by default. These labels generally look better than the simple variable names, but there may be times when you want a more utilitarian type of report that shows the variable names instead of the variable labels. You can get this type of report with the USE VARIABLE NAME statement. Wherever it is applied, the columns will headed by variable names.

Level USE VARIABLE NAME can be specified for individual variables or columns at the report level.

Default

```
USE VARIABLE LABELS;
```

Reports are formatted using variable labels.

Example

```
FOR REPORT ONE:  USE VARIABLE NAMES;
```

Effect For Report One, all columns are headed by variable names rather than labels.

APPENDIX A

Installation (Windows)

HOW TO INSTALL TPL REPORT UNDER WINDOWS

Note to TPL TABLES Users

Installation of TPL REPORT is similar to installation of TPL TABLES. If you have both TPL TABLES and TPL REPORT, and they have the same version numbers, they can be installed in the same system directory.

If you have TPL TABLES and TPL REPORT on the same CD, TPL REPORT will automatically be installed in the same directory as TPL TABLES.

Shared profile.tpl

If TPL REPORT and TPL TABLES are installed in the same directory, the two applications will be sharing the same **profile.tpl**. In this case, you should be aware of the fact that statements entered in the profile will be shared by the two applications. In addition, the words TABLE and REPORT are equivalent. For example, the FORMAT statement ALIGN TABLES LEFT; will align reports to the left as well.

Any FORMAT statement that applies only to TPL REPORT will be ignored by TPL TABLES. The reverse is also true, so there is no problem with statements that apply to only one of the applications.

There is a small possibility that you could put a FORMAT statement in the system profile that is meaningful in both applications but has different results. Or, you might wish to set a default for one application that is inappropriate for the other. In the unlikely event that this occurs, we recommend that you make a new copy of the profile in the directory where you are running your report job(s).

Installing from the CD

If you are replacing an earlier version, please review the next section before installing the new version.

To install, insert the CD in the CD drive. After a pause, the installation process may begin automatically. If it does not start automatically, go to **Start** then **Run**. Select the file **setup.exe** on the CD and click on "OK".

Respond to the prompts.

If You Have an Earlier Version of TPL REPORT

.tpl Files

During installation, new **profile.tpl**, **color.tpl** and **country.tpl** files will be placed in the TPL system directory. If you are installing this version on top of a previous one and have previously edited these files to establish your own set of system defaults, you will probably want to save them in another place and copy them into the system directory after doing the new installation.

Replacing a Previous Version

If you do not want to retain your previous version and you want to install the new version in the same place as the old, we recommend that you uninstall the old version before installing the new one.

Using More than One Version of TPL REPORT

A new version of TPL REPORT can be installed without removing earlier versions. The different versions will not interfere with each other provided that they are installed in different directories and run against codebooks processed by the correct version. The **tpl.ini** file, described below, can contain path information for multiple versions of TPL REPORT. Preference settings will be shared by the versions.

tpl.ini

For Version 6, a file named **tpl.ini** was installed in the Windows system directory (e.g. c:\winnt or c:\windows). **tpl.ini** is a text file that contains information about your TPL preferences and also path information for TPL modules. Each time you use TPL, the current preference settings will be saved in **tpl.ini**. You should not directly modify this file. Instead use the various preferences menus in the TPL system to set these values. For Version 7, the **tpl.ini** file has been moved to a location specified by the environment variable **TPL_INI.** This environment variable

is set during installation. Version 6 will continue to use the copy of **tpl.ini** in the Windows system directory. So if you have both Version 6 and Version 7 installed, they will not necessarily have the same preferences.

Network Installation

In a network installation, it is desirable to have the TPL programs in a common location on a server but make preferences user specific. Version 7 supports this goal in such a way that it does not compromise restricted directories. TPL Version 7 uses two environment variables, **TPLPATH7.0** and **TPL_INI.** On each machine using TPL, set TPLPATH7.0 to the common server location where the modules are installed. Set TPL_INI to a location on the user's machine that he has read and write access to.

It is also desirable to add a directory structure to the user's start menu with entries for each of the programs, documentation, and help files included in a standard install. Adding a desktop shortcut which points to this directory structure in the start menu completes a full network install.

Compatibility

"Source" Files

"Source" files, including codebook sources, report requests, and format requests, that run with earlier versions of TPL REPORT should run without change. The only exception is if you have a name that has been added to the list of Keywords. In the unlikely event that this happens, you will get an message and will need to change the name.

Codebooks and TPLR Subdirectories

Codebook objects (processed codebooks) created by earlier versions are not compatible with Version 7.0. Before running report jobs, you will need to reprocess the codebooks.

TPLR subdirectories created by earlier versions of TPL REPORT are not compatible with Version 7.0. You cannot do a rerun process using an old TPLR subdirectory with this version of the system.

Default Settings in Profile.tpl

After you have installed TPL REPORT, there will be a file called **profile.tpl** in the directory where you installed the system. It contains a set of text statements that determine defaults for basic activities. A sample profile after installation is:

```
Postscript = yes ;
Default font = H 8;
Footnote text font = T 8;
Footnote symbol font = H 8;
Title font = HB 10;
paper = LETTER;
```

All statements entered in the profile during installation are described in detail in the FORMAT Language chapter of the manual.

If you wish, you can change the values in the profile after installation and also have different profiles for different sets of jobs. The initial settings assume that you will want to work in PostScript mode and set default fonts. PostScript provides the best display formats. You can view PostScript reports in Ted, the TPL Editor, and you can print them from Ted, regardless of whether or not you have a PostScript printer.

If you do not want to be in Postscript mode, you can delete the Postscript line or change "yes" to "no". The font statements are ignored in non-PostScript mode.

Networks

Licensing Note

If you are accessing a copy of TPL REPORT installed on a network server, you must have a license to use TPL REPORT on your PC.

APPENDIX B

Run Instructions (Windows)

INSTRUCTIONS FOR RUNNING TPL REPORT UNDER WINDOWS

Introduction

TPL REPORT can be run using interactive menus, or it can be run as a batch process using scripts. Scripts are described in a separate appendix as well as in TPL Help. This appendix describes the basic information needed to run jobs from menus and the various input and output files for different types of jobs. See also TPL Help for additional details about the drop-down menu options in the interactive menus for running jobs.

TED and Other Editors

TPL REPORT is designed to allow you to use the editor (word processing program) of your choice to create codebooks, report requests and format requests. Any editor that creates stand-alone ASCII text files is acceptable.

You can also use TED, the TPL Editor. TED lets you edit, view, and print character files. It also allows you to view and print Postscipt reports produced by TPL REPORT. You can use it to print Postscript reports on any Windows compatible printer even if the printer does not support Postscript. Finally TED allows you to export reports in formats usable by other software.

If you are running a job that stops because of errors, TPL REPORT will transfer to TED to allow you to view the error messages and make corrections. When you are finished, you can return to TPL REPORT to resume processing.

TED is an integral part of TPL REPORT, but you can also use it without starting TPL REPORT. See TED Help for complete details.

Description of Jobs and Files

Getting Started

You can start TPL by clicking on the TPL icon or by going to **Start** then **Programs** then **QQQ Software** then **TPL**.

You can run as many jobs as you wish without leaving TPL.

Selecting the Job Directory

The **Job Directory** is the directory in which your TPL jobs will run. You will probably find it most convenient to set the Job Directory to the directory where your codebook, data and request files are stored, but you can choose a different directory if you wish.

Outputs are stored in the Job Directory. These outputs include the processed codebooks and TPL subdirectories described in this appendix. For this reason, it is important to know what Job Directory you are in. If an output is not found in the expected place, the likely reason is that the Job Directory was set to a place other than the intended one.

To see what the current Job Directory is or to change to a different Job Directory, go to **File** then **Job Directory**.

Creating and Processing Codebooks

Codebooks can be created either with an editor or interactively with Codebook Builder. To create a codebook interactively, go to **File** then **Build Codebook** in the main TPL screen. Instructions for creating codebooks interactively can be found in ***Codebook Builder Help***.

After you have created the codebook, you can save it into a file with a name of your choice. Usually the codebook is saved into a file with the same name used at the beginning of the codebook. For example, if you name the codebook **Survey** with the codebook entry **Begin Survey Codebook**, save it with the name **Survey.cbk**. The codebook file you have created is called the **codebook source**.

Run the codebook processor, giving it the name of the codebook source. In the main TPL screen, go to **Run** then **Codebook**. When prompted, enter the name of your codebook source file, for example **Survey.CBK**.

If any errors are found in your codebook, you will be transferred to TED where you will see two windows open, one containing your codebook source and another showing the source with error messages. Edit the codebook source to correct the errors. When you are finished, Go to **Return to TPL** then **Save changes and try again**. Your corrected codebook source will automatically be saved before processing continues.

Note In some cases, you may not wish to resume processing. For example, if you have accidentally entered in the menu the name of a file that is not a codebook source file, you will need to go back to the menu to correct it. In this case, go to **Return to TPL** then **Cancel**.

As the codebook is being processed, the source and any error messages are saved in a file with the same name as the codebook and a **.O** extension. For example, if your codebook is named Survey, the file is called **Survey.O**.

When your codebook has been processed successfully with no errors, the **.O** file will be deleted and the **.L** codebook abstract file will take its place. You can view and/or print the abstract in TED by clicking on the **Review/Print** button when your job is completed. When you are finished, you can close TED or go to **Return to TPL** then **Resume**.

Codebook Abstract

The abstract includes the name of the codebook source file, the date and time of processing, and the TPL version number. In addition, it contains a list of the codebook variables in alphabetical order along with each variable's size and location within a record. This information is particularly useful if you have an alignment problem between your codebook and your data file. You may also find the abstract useful as a quick reference when preparing your report specifications. If you are creating a codebook describing a **CSV** or other type of delimited file or a database, the information in the abstract will differ slightly.

For each control variable, there is a count of the number of condition values.

Note If your data file is an ASCII file, it will have carriage return/line feed characters at the end of each record. These will not be included in the record sizes listed in the abstract. ASCII is the default file type.

Codebook Object

The processed codebook is called the **codebook object**. When you have successfully run the codebook processor, your codebook object will be stored with **.K** appended to the name. Thus, for a codebook named Survey, the codebook object will be given the name **Survey.K.**

Once your codebook is successfully processed, you can run any number of report jobs using the same codebook object.

Note The name of your codebook object will always be derived from the name you have used in the BEGIN codebookname .... entry in the codebook, regardless of the name you give to your codebook source file.

Database Codebook Source

The following applies if you have the TPL-SQL database interface. When a database codebook is processed, there is another file generated in addition to the **.K** and **.L** files. This is the **.S** file. When a you create a codebook source for a database, you omit some items such as field widths and control variable condition values. These are filled in by gathering data from the database. The **.S** file is a new codebook source with the additional data filled in. See the TPL-SQL manual and/or TPL Help for more details.

Producing Reports

Report requests can be created with Ted or another editor of your choice.

In the USE statement at the beginning of a report request, you can refer to the codebook using the same name you used in the **BEGIN codebookname CODEBOOK** statement. Using the name **Survey** shown in the example above, you would say **USE Survey CODEBOOK;** at the beginning of your report request. TPL REPORT will know to look for a codebook object file called **Survey.K** for descriptive information about your data file.

If your codebook is in a subdirectory other than the one in which you are running your report job, you can give the complete name for the codebook in the USE statement.

Save your report request with any valid Windows file name, for example, **Survey.REQ.**

You may also have an optional format request giving detailed specifications for formatting your reports. The format request can have any valid Windows file name, for example, **Survey.FMT**.

To produce reports, you need to enter the names of the report request file, the data file and (optionally) the format request. In the main TPL screen, go to **Run** then **Report Request**. When prompted, enter the name of your report request, your data file, and (optionally) the format request. If your data is contained in more than one file, see the Data chapter for instructions on multi-file input.

If any errors are found in your report or format requests, you will be transferred to TED where you will see two windows open, one containing your report request or format request and another showing the output with error messages. Edit the request to correct the errors. When you are finished, go to **Return to TPL** then **Save changes and try again**. Your corrected request will automatically be saved before processing continues.

Note

In some cases, you may not wish to resume processing. For example, if you have entered an incorrect data file name in the menu, you will need to go back to the menu to correct it. In this case, go to **Return to TPL** then **Cancel**.

When TPL REPORT has finished processing your data and creating your reports, you can review the reports and other output on the screen, print them, or export the reports into files of different types such as Encapsulated PostScript. To do this, transfer to TED by clicking on the **Edit/Print** button. Your outputs will be opened in TED. When you are finished, you can close TED or go to **Return to TPL** then **Resume**.

The TPLR Subdirectory

Each time you run a report job, a subdirectory is created to contain the files needed to create your reports. The subdirectory has the name **TPLRnnnn** where **nnnn** is a randomly generated number with 1 to 4 digits. You can override this number by entering the number of your choice in the Report Request screen where you enter the file names to be used for your report job.

See also the Script arguments -O and -N for selecting subdirectory numbers in TPL scripts.

Some of the files in the TPLR subdirectory are illegible from your point of view. The important files are those that contain the completed reports and the file named OUTPUT.

The Report Files

Multiple REPORT statements can be included in the same job. A separate report file is created *for each* REPORT statement. A report file name begins with the **report-name** assigned in the REPORT statement and ends with the suffix **.rep** . For example, if the report-name is IND_REPT, the report is saved in a file called **ind_rept.rep**

If **Postscript = yes**; is specified, the suffix for the report files is **.ps**, for example **ind_rept.ps** .

The OUTPUT File

The **OUTPUT** file contains your request files, the names of your data and request files, the date and time of execution for each part of the job, the TPL version number and, at the end, the number of rows in each report.

Subdirectory Maintenance

If you go to Run then Remove Directories in the main TPL screen, you can get a list of TPLR subdirectories. In addition to removing all or selected directories, you can add notes to the directories. If you click on a subdirectory on the list, you will get the date and time that the subdirectory was created and a display of any notes that have been added.

The subdirectories on the list are those contained in the current Job Directory. The Change button lets you change to a different Job Directory.

Note If you have run TPL TABLES jobs in the selected Job Directory, the TPL subdirectories for the tables jobs will also be listed.

Customizing with PROFILE.TPL

The TPL software installation process creates a file called **PROFILE.TPL** and puts it in the TPL system directory. PROFILE.TPL is a text file that you can edit. It contains FORMAT statements that become defaults for such things as PostScript fonts and paper size. You can change these defaults and also add other FORMAT statements to set additional defaults. For example, if you always want your reports left-adjusted on the page, you can make it the default by including the FORMAT statement ALIGN REPORTS LEFT; in PROFILE.TPL.

If you want to leave the system profile unchanged, but use a different profile for a particular set of jobs, you can make a copy of PROFILE.TPL in the directory where you are working and change that copy to fit the reports you are preparing. The profile in the directory where you are working will override the one in the TPL system directory.

Shared profile.tpl

If TPL REPORT and TPL TABLES are installed in the same directory, the two applications will be sharing the same **profile.tpl**. In this case, you should be aware of the fact that statements entered in the profile will be shared by the two applications. In addition, the words TABLE and REPORT are equivalent. For example, the FORMAT statement ALIGN TABLES LEFT; will align reports to the left as well.

Any FORMAT statement that applies only to TPL REPORT will be ignored by TPL TABLES. The reverse is also true, so there is no problem with statements that apply to only one of the applications.

There is a small possibility that you could put a FORMAT statement in the system profile that is meaningful in both applications but has different results. Or, you might wish to set a default for one application that is inappropriate for the other. In the unlikely event that this occurs, we recommend that you make a new copy of the profile in the directory where you are running your report job(s).

Encapsulated PostScript (EPS)

Most desktop publishing software that allows you to add PostScript files to a document requires that these files follow certain conventions. Files that follow these conventions are called Encapsulated PostScript (EPS) files. The **EPS** files created by TPL REPORT can be incorporated into your document according to your desktop publishing software's rules for bringing in Encapsulated PostScript files. EPS files have the file extension **.eps**.

There are three ways of converting PostScript reports to EPS.

1. If a PostScript report is open in TED, you can export EPS interactively. There will be one **.eps** file for each page of report output. The files will be named with the report name and the page number. For example, if a report has 6 pages and the PostScript file is named **ind_rpt.ps**, the .eps files will be named **ind_rpt1.eps, ind_rpt2.eps,** ....., **ind_rpt6.eps**.

2. EPS files can be exported using TED arguments in scripts as described both in the Scripts appendix.

3. The ENCAPS program provides a third way.

ENCAPS

ENCAPS is a stand-alone command line program that is installed in the TPL system directory. To run it, change into the TPL subdirectory that contains the **.ps** file you wish to convert. Assuming that TPL REPORT in installed in C:\QQQ\REPORT, give the command:

```
C:\QQQ\REPORT\WTPL\ENCAPS  .  0  report-name.ps  <Enter>
```

The report will then be divided into pages and an **EPS** file will be created for each page. These files will be saved in the TPL subdirectory and will be named according to the page number. For example, if you have a three page report, it will be divided into three files with the following names:

```
P1R1.EPS
P2R1.EPS
P3R1.EPS
```

ENCAPS will report the names of the **EPS** files as they are created.

Other options are available, such as naming the directory for the output instead of specifying '.' for the current directory and running silently with no reporting (1 instead of 0). The current options will be displayed on the screen if you type:

```
C:\QQQ\REPORT\WTPL\ENCAPS  <Enter>
```

Exporting CSV (delimited) files

You can also create CSV files from PostScript reports. CSV files have the extension **.csv**.

There are two ways of doing this.

1. You can export the files interactively from TED after doing a report run using the Run menus. See TED Help for details.

2. You can export the files using TED arguments in scripts as described both in TPL Help and in the Scripts appendix.

Common Error Messages

Error messages are intended to be self-explanatory. However, two common errors deserve special note.

Common error message 1

```
*** ERROR: A syntax error was discovered while processing  'element'.
        Look for the error at or before that point.
```

This message appears whenever there is a syntax error in a codebook, report request, format request or profile. Examples of syntax errors are misspelled keywords or punctuation errors such as a missing colon (**:**) or semicolon (**;**). The point at which TPL REPORT discovered the error is indicated by the element in quotes.

Example The following sequence in a report request will produce the message shown below.

```
REPORT ONE 'Dispatch Report, Sorted by Date and Service Type'
        DISPATCHES THEN MONTH THEN DAY THEN CASE_NO
        THEN SERV THEN DISP THEN SQUAD;

*** ERROR:  A syntax error was discovered while processing
'DISPATCHES'.  Look for the error at or before that point.
```

Since the error was found when the name DISPATCHES was encountered, we can assume that there is something wrong with the word DISPATCHES, or that an error preceded the word DISPATCHES. In this example, a colon (:) is missing following the report title. TPL REPORT is looking for the colon when it finds the word DISPATCHES.

Common error message 2

```
*** ERROR: The variable 'variable name' is undefined.
```

A frequent cause of this error is a misspelled name. Another cause is a reference to a variable that has not yet been defined. For example, if a variable is computed in a COMPUTE statement and used in a REPORT statement that precedes the COMPUTE statement, the computed variable is unknown to TPL REPORT when it finds it in the REPORT statement.

Example Misspelling of the variable name INCOME as INCOM produces the message shown below.

```
SELECT IF INCOM > 50000;

*** ERROR:  The variable 'INCOM' is undefined.
```

Networks

Licensing Note

If you are accessing TPL REPORT on a PC connected to a network, you must have a license to use TPL REPORT on your PC.

APPENDIX C

Scripts (Windows)

RUNNING BATCH JOBS WITH TPL SCRIPTS

Introduction

TPL jobs can be run from the character mode command line, the **Run** command found in **Start**, or a batch file. It is also possible to create a script which allows multiple TPL and non-TPL jobs to run without your intervention. To start TPL in one of these ways, run the program **WTPL**.

To run an individual job, you just type the command with all required command-line arguments. If a required argument is missing, menus will open prompting you for the missing data. At the end of the job, all menus will close and the job will terminate. For example, assuming that TPL REPORT is installed in c:\qqq\table, suppose you type in the Run menu:

```
c:\qqq\table\wtpl codebook -p c:\qqq\table\examples -c cps.cbk
```

WTPL will process the **cps.cbk** source found in the examples directory. If you omitted the **-c cps.cbk**, a menu would prompt you for the name of the codebook to be processed.

If you wish to run a collection of jobs, you can start WTPL with a file name containing a script consisting of a list of the commands you wish to execute. The scripts may include substitution arguments. The values of these are placed on the command line. Again if you omit required arguments and the job is run in foreground, the system will prompt you. When one command has completed, the next in the script will execute without calling TED or prompting for TED. This will continue until the script is exhausted.

Notes

The command arguments are case-sensitive.

Exactly one command and its arguments can appear on each line of a script. There is no way to continue a line and you cannot put multiple commands on a line.

A line that is completely blank will terminate the script so that nothing following the blank line will be executed.

There is no facility for conditional execution.

If a job is run in foreground, an error in a request will put you in TED. When you have corrected the error the script processing will resume. An error in the script itself will usually result in that portion of the script being skipped. If you just omit an argument or enter an incorrect one, you will usually be put in a menu to fill in the missing information.

Paths and arguments including blanks are supported but such items must be in quotes. ***They must be double, not single, quotes.*** For example using the Run command described below you might have:

```
RUN "my programs\program.exe" 1 xxx "new arg"
```

WTPL has a startup directory which may be changed from within the program using the **Job Directory** option under **File** and saved using the **Save Job Directory** option under **Preferences**. However, if you do not run all of your TPL jobs from the same directory, it is easier to include a **CHDIR** command as the first entry in each of your scripts. This will make it unnecessary to include full path names for all files referenced in your scripts.

Files and directories may use absolute (full) paths or relative paths. Paths are relative to the most recent CHDIR command. For example:

```
CHDIR c:\qqq\table\examples
TED -Pp tplr1\rep_one.ps
```

will print the same file as

```
TED -Pp c:\qqq\table\examples\tplr1\rep_one.ps
```

Job Script Example

The following sample script runs several jobs, re-using the **TPL1** subdirectory after copying the PostScript **.ps** report output to another location. Note that all of the job files are in the same directory. Starting the script with a **chdir** to that location means that full path names are not required for job files. The TPL system is assumed to be located in **c:\qqq\table**.

Start the script from the command line or using the **Run** option of **Start** by entering:

```
c:\qqq\table\wtpl -A c:\qqq\table\examples\sample.lst
```

The **sample.lst** file is:

```
chdir c:\qqq\table\examples
mkdir myreports
codebook -c cps.cbk
report -r cps1.rep -d cps.dat -f cps1.fmt -O tplr1
copy tplr1\cps1.ps myreports\cps1.ps
report -r cps2.rep -d cps.dat -f cps2.fmt -O tplr1
copy tplr1\cps2.ps myreports\cps2.ps
report -r cps3.rep -d cps.dat -f cps3.fmt -O tplr1
copy tplr1\cps3.ps myreports\cps3.ps
codebook -c police.cbk
report -r police.rep -d police.dat -f police.fmt -O tplr1
copy tplr1\police.ps myreports\police.ps
```

Wild Cards (* and ?) in TED, COPY, and DELETE Commands

File name arguments in TED, COPY, and DELETE can include the * and ? wild cards.

The * wild card can take the place of 0 or more characters. For example, if PostScript reports have been collected in a single directory with different names followed by the extension **.ps**, they can all be printed by the following Ted command.

```
TED -pP *.ps
```

The ? wild card can take the place of exactly one character. For example, if a report has been converted to multiple EPS report files, one for each page, the command:

```
TED -pP P?R1.eps
```

will print the files P1R1.eps, P2R1.eps, ... P9R1.eps. It will not however print files with names such as P10R1.eps or P25R1.eps, because a match would require more than one character.

The same wild cards can be used to copy files to another directory and to delete files. In the following example, all .eps files are copied from the current directory to the directory e:\my_eps_files. Then the .eps files are deleted from the current directory.

```
COPY *.eps e:\my_eps_files
DELETE *.eps
```

Running a Script in Foreground or Background

If the line invoking a script begins with **-A** the script is run in foreground. If the line begins with **-B**, the script is run in background.

A script run in foreground shows the progress of the steps of the script. For example, as the data is read, the hourglass shows its progress. If a required argument is omitted or is incorrect, the system displays a prompt for the argument and processing is stopped until the argument is provided. If a request error is detected, you are put into Ted for editing just as if your were running the job interactively.

A script run in background behaves quite differently. There is no activity shown on the screen except for an icon at the bottom. The icon changes to reflect the approximate percent of the script completed. If any error occurs, the script is terminated. It is recommended that you use a Script Log for background scripts.

Script Log

A Script log is a brief listing of the results of the steps of a script. It is created when it is specified on the command line for the script. The script log specification is an optional parameter **-G** followed by the name you choose for the Script log file. If it appears, it must follow the **-B script-name** or **-A script-name** and must precede any substitution arguments.

The resulting log has 1 line for each Report, Codebook, TED, Table, or Rerun step in a script. The first word in each line is one of the following:

SUCCESS:
FAIL:
WARNING:
ERROR:

The line will also contain the name of the program being run and the script line number. If the line of the script fails, the log will contain either an error message or the comment to look in the output file for more details.

The script log is useful for debugging scripts. It is also useful for programs which include TPL scripts. You can check the success of a job by reading the first character of each line of the script. If all lines begin with **S** or **W**, then the TPL jobs in the script executed successfully.

A **WARNING** line can appear when a format request has warnings. These can usually be ignored. The **output** file messages associated with these format request warnings are preceded by ***** NOTE**.

Some other things that can cause a **WARNING** are: data errors are found when the data is being read; or the job runs successfully, but there is no data for the report(s), for example because a Select statement caused no data to be selected for the report(s).

Example c:\qqq\table\wtpl.exe -B c:\myfiles\myscript.lst -G c:\myfiles\mylogfile

Substitutions in Scripts

A list of one or more substitution arguments may be added to the **-A** or **-B** command line following the script name. These replace the items in the script referenced by ***%1***, ***%2***, etc. For example, if the command line is:

wtpl -A sample.lst T1 F1

and the first line of **sample.lst** is:

report -r %1.req -d cps.dat -f %2.fmt -O tplr1

The result is:

report -r T1.req -d cps.dat -f F1.fmt -O tplr1

Example Using Substitution Arguments

The script in the earlier example can be modified to use substitution arguments. The new command line is:

```
c:\qqq\table\wtpl -A c:\qqq\table\examples\sample.lst cps1 cps2 cps3
```

The substitution arguments follow the script name and are referenced in order as %1, %2, etc.

The new script is:

```
chdir c:\qqq\table\examples
mkdir myreports
codebook -c cps.cbk
report -r %1.rep -d cps.dat -f %1.fmt -O tplr1
copy tplr1\%1.ps myreports\%1.ps
report -r %2.rep -d cps.dat -f %2.fmt -O tplr1
copy tplr1\%2.ps myreports\%2.ps
report -r %3.rep -d cps.dat -f %3.fmt -O tplr1
copy tplr1\%3.ps myreports\%3.ps
codebook -c police.cbk
report -r police.rep -d police.dat -f police.fmt -O tplr1
copy tplr1\police.ps myreports\police.ps
```

Commands and Arguments

WTPL Arguments for Starting Scripts

```
-A script-name [run script in foreground]
-A script-name substitution-argument-1 substitution-argument-2...
-A script-name -G log-file substitution-argument-1 substitution-argument-2...
-B script-name  [run script in background (as icon)]
-B script-name substitution-argument-1 substitution-argument-2 ...
-B script-name -G logfile substitution-argument-1 substitution-argument-2 ...
```

Script Commands and Arguments

REPORT (or report)

-p working-directory
working-directory is either the path to the directory where you want the job to run or the word DEFAULT to indicate the current job directory. A period (.) can be used in place of the word DEFAULT. **-p** is most useful for submitting a request for a single job on the command line. For a command in a script, chdir is more convenient.

-r request [REQUIRED]

-d data-file [REQUIRED except as noted below]
Instead of using the **-d** argument to specify a data file, you may use **-l** with the name of a file whose contents is a list of data files or you may use ODBC Database arguments.

-l file-list
Note: File lists are described in the "Data" chapter.

-f format-request

-O old-run-directory
Use subdirectory **nnnn** and overlay its contents if it already exists.

-N new-run-directory
Create a new sudirectory with the number **nnnn** only if there is not already a subdirectory in your current directory with the name **TPLRnnnn**. If such a subdirectory already exists, TPL REPORT will not use it but will instead create a new subdirectory with a random number.

Note **TPLR** can precede the number in the -O and -N arguments. An example is **-O TPLR25**. If you do not precede the number with **TPLR**, **TPL** will be used in the subdirectory name. For example, with an argument of **-O 25**, the subdirectory **TPL25** will be created.

ODBC Database Arguments

If you have the TPL-SQL interface for ODBC, you can use the following arguments with the TPL commands: REPORT, CODEBOOK, RERUN, and RMTPL. Normally they would only be used with REPORT or CODEBOOK. For more information, see the section on Arguments for ODBC.

-q [Use **-q** or **-Q** instead of **-d** when using an ODBC Data Source.
If -q is used, TPL REPORT may prompt for the ODBC Data Source.]

-Q ODBC-datasource-name

-U database-user

-P database-password

CODEBOOK (or codebook)

-p working-directory
working-directory is either the path to the directory where you want the job to run or the word DEFAULT to indicate the current job directory. A period (.) can be used in place of the word DEFAULT. **-p** is most useful for submitting a request for a single job on the command line. For a command in a script, chdir is more convenient.

-c codebook-source [REQUIRED]
codebook-source is the name of the codebook source file.

CBUILDER (or cbuilder) - ODBC Databases only

This command lets you call Codebook Builder from a script to update condition value lists if your database has changed since you last created the codebook. New values are added at the ends of the condition value lists.

-u [REQUIRED]

-K codebook-object.K [REQUIRED]
Provide the complete name of the codebook object ("old" processed codebook) to be used as input. The **.K** extension must be included in the name.

-c updated-codebook-source [REQUIRED]

-Q ODBC-datasource-name [REQUIRED]

If you need a user name and password to access your database, you can add the following arguments to avoid being prompted for the information.

-U database-user

-P database-password

CBUILDER does not have an argument for working-directory. Thus, if you are using this command as a stand-alone, you may need to include full path information for the codebook names. If you are using the command in a script, you can precede it with a CHDIR command to get to the directory where you want to update the codebook.

Example

```
CHDIR f:\myjobs
CBUILDER -u -K survey.K -c survey_new.cbk -Q "Survey Data"
```

RMTPL (or rmtpl)

-p working-directory

-X jobs-to-delete [nnnn or TPLRnnnn or TPLnnnnn or full path [REQUIRED]

-X ALL (may be used instead of the above)

RUN command-and-args

RUN can take any executable and its arguments but not "built-ins" such as dir or copy. It also cannot take > or <.

CALL command-and-args

CALL can take any executable that RUN can take and also supports "built-ins" such as dir and copy. It supports redirections > | and <. Unfortunately, in some Windows operating systems, one CALL command may execute before the previous one completes. So be careful using this command.

CHDIR path

CHDIR supports changing to any existing path including ones on different drives. If most of your job files are in the same directory, you will probably want to include a **CHDIR** command as the first entry in your script so that you do not have to provide full path names for all files referenced in the script.

MKDIR path

In TPL scripts MKDIR can make a path more than 1 segment at a time.

MOVE old-name new-name

MOVE allows you to move a file from one directory to another.

REM any-text [no action performed]

REM can be used to add comment lines to a script.

COPY

COPY from-file to-file

COPY from-file(s) to-directory

In COPY file(s) to directory, the from-file(s) argument can include the * and ? wild cards. Wild cards are explained elsewhere in this appendix.

DELETE file(s)

DELETE arguments can include the * and ? wild cards. Wild cards are explained elsewhere in this appendix.

TPLDIR reference-name

TPLDIR is described in the TPLDIR section of this appendix.

TED

TED arguments can include the * and ? wild cards. Wild cards are explained elsewhere in this appendix.

-e ascii-file-to-review
-p Postscript-file-to-review
-eP ascii-file-to-print
-pP Postscript-file-to-print
-pE Postscript-file-to-convert-to-eps
-pF Postscript-file-to-convert-to-PDF
The entire file is converted and placed in the same directory as the source with the same name except **ps** is changed to **pdf**.
-pV Postscript-file-to-convert-to-CSV
-pV can be followed by a divider character to be used in place of comma to separate the values in the exported file(s). If you want to use a blank, enclose the entire argument in quotes: "-pV ". Note: Tab cannot be specified in a script. If you are exporting interactively from TED, you can select Tab as the divider. You can also use the CSV DIVIDE format statement to specify the divide character that will be used for Unix or Windows.
-D Export-directory
The TED Export-directory is described in another section of this appendix.
-N Export-core-name
The TED Export-core-name is described in another section of this appendix.

The **-e** and **-p** arguments will stop the processing stream to enable you to review the reports and output file. You may run several TPL REPORT jobs and then review all of the reports and output at once using TED with multiple **-e** and **-p** arguments. If you use **-eP** or **-pP**, TED will be invoked, the files will be printed, and TED will close without any human action. **-pE** will also do its task without stopping the processing.

Notes on Export to CSV

When TED converts a PostScript file to CSV, the PostScript file must be in the TPLnnnn subdirectory where it was created. TED uses other files in the subdirectory to do the conversion and will not be able to find them if the PostScript file has been moved to a different location.

Setting the TED Export Directory in Scripts

By default, exported files are placed in the same directory as the source **.ps** file. When you export interactively in TED, you can change this destination. The **-D** argument allows you to change the export directory in a TPL script.

To specify an export directory, place **-D** *export-directory* on a TED line before the **-p** export argument. The new export directory remains in effect until the end of the script or until there is another **-D** argument. To return to the default behavior specify **-D DEFAULT**.

Export Core Name in Scripts

When PostScript files are exported, they are divided into a number of files equal to the number of report pages. The file names for the exported files consist of three parts: an **export directory**, a **core name**, and an **extension**. For Encapsulated PostScript, the extension is **.eps** and for bit mapped graphics it is **.bmp**.

When files are exported from TED interactively, the default core name is always **report-name***n* where *n* is the page number.

Encapsulated PostScript can also be exported by TED script commands. When files are exported with a script command, the default core name varies depending on the export type. For Encapsulated PostScript, the default core name is **P***n***T***m* where *n* is the page number and *m* is the report number.

To specify a different export core name, place **-N** *core-name* on a **TED** line ***before*** the **-p** export argument. The new export core name remains in effect until the end of the script or until there is another **-N** argument. To return to the default behavior specify **-N DEFAULT**.

Example For a report request with 5 report pages and a **one.ps** file in **TPLR2**, the following script will export .eps files with a core name of **salary**. The .eps files will be named **salary1.eps, salary2.eps, ..., salary5.eps**.

```
CHDIR TPLR2
TED -N salary -pE one.ps
```

Example For a report request with 5 report pages and a **one.ps** file in **TPLR2**, the following script will export .eps files with a core name of **Report**. The .eps files will be named **Report1.eps, Report2.eps, ..., Report5.eps**.

```
CHDIR TPLR2
TED -N Report -pE one.ps
```

Note You can also create Encapsulated Postscript using the ENCAPS command line program described elsewhere in this manual. The ENCAPS program uses the core name **P***n***R***m*.

Note If you have more than one report on a page, they will all be contained in the same **.eps** file.

TPLDIR Script Command

When a report job is run, a TPLRnnnn (or TPLnnnnn) directory is created. This directory contains the finished reports and the output file. When operating interactively, you may select a specific TPLRnnnn directory or allow the system to select a unique name. In a script, if you use a specific name, you run the risk that some other job might have used that directory name. If you let the system select the name, you have no way of doing additional things with the directory. For example you can't use TED to print the reports or convert them into EPS since you don't know what the directory name is.

The TPLDIR command solves this problem. TPLDIR creates a unique TPLnnnnn subdirectory in the currently active directory and associates it with a user-selected *reference-name*. The script can then reference the directory by using *%reference-name*.

Note When TPLDIR is used, the assigned directory name will begin with **TPL**, not **TPLR**.

Example

```
CHDIR c:\test
TPLDIR cpsjob
TPLDIR dispatchjob
REPORT -r cps.rep -d cps.dat -f  cps.fmt -O %cpsjob
REPORT -r dispatch.rep -d dispatch.dat -f dispatch.fmt -O %dispatchjob
TED -pP c:\test\%cpsjob\cps.ps
TED -pE c:\test\%dispatchjob\dispatch.ps
```

This script will run the cps and dispatch report requests. It will print the reports produced by the cps job and convert the dispatch reports into EPS.

Note that in the report jobs, we used **-O** for old directory rather than **-N** for new directory since the directories were actually created by the TPLDIR command. Also note that the CHDIR command occurs before the TPLDIR commands. Otherwise the directories created by TPLDIR might be in the wrong place.

Arguments for ODBC

If you have the TPL-SQL interface for ODBC, you can use the following arguments to access ODBC Data Sources from scripts.

```
-q    [If -q is used, TPL REPORT prompts for the ODBC Data Source.]
-Q    ODBC-datasource-name
-U    database-user
-P    database-password
```

Arguments which have blanks or special characters must be put in quotes. They must be double, not single, quotes.

Depending on your environment, you may or may not be required to provide a user name and password to access the Data Source. If you do not wish to include a database user name and password in your script, you may use substitution arguments for these parameters and then provide the user name and password when you run the script.

If you provide all required arguments, you can run your request without being prompted for any information about your ODBC Data Source.

The -q argument can be used if you wish to continue with the same database. You can enter a new -Q and other arguments if you wish to change databases.

Example

In the following sample script, a codebook will be processed for the ODBC Data Source named "My datasource", a report request will be run using the data from the same Data Source, and a second report request will be run using data from a different Data Source. No prompts will be needed for Data Source.

```
CODEBOOK  -c my_db.cbk -Q "My datasource"  -P xxx  -U "John Doe"
REPORT -q  -r sample.req
REPORT -Q "my other datasource"  -P yyy  -U sew  -r another.req
```

Notes

For codebook processing, ODBC Data Source arguments are only required if the codebook needs information from the database. For example, if an ODBC codebook is created interactively in Codebook Builder, all required database information will already be included in the codebook source.

APPENDIX D

Installation (UNIX)

HOW TO INSTALL TPL REPORT UNDER UNIX

How to Stop

You can stop the **setup** procedure by entering **<Ctrl>C.**

Before You Start

The TPL REPORT installation process copies TPL REPORT to your hard disk. It also asks you about certain characteristics of your operating environment, such as printer, so that it can set defaults for system operation. We recommend that you scan through the following instructions before you start, so that you will know in advance how you want to answer the installation questions.

If you wish to move the TPL REPORT system to another location in your file system after it is installed, you must remove it from the original location and reinstall it. Merely copying the files will not work correctly. If you have customized your TPL REPORT **profile.tpl**, **color.tpl** or **country.tpl** files, you may wish to save them for use in the new location before removing TPL REPORT from the previous location.

Note to TPL TABLES Users

Installation of TPL REPORT is similar to installation of TPL TABLES. The installation program, called **setup**, gives you the option of installing TPL TABLES, TPL REPORT, or both at the same time.

If you have both TPL TABLES and TPL REPORT, and they have the same version numbers, they can be installed in the same system directory. The **setup** program will ask you if you wish to do this and will check to insure compatibility of the versions if you request installation in the same directory.

Shared profile.tpl

If you install TPL REPORT and TPL TABLES in the same directory, the two applications will be sharing the same **profile.tpl**. In this case, you should be aware of the fact that statements entered in the profile will be shared by the two applications. In addition, the words TABLE and REPORT are equivalent. For example, the FORMAT statement ALIGN TABLES LEFT; will align reports to the left as well.

Any FORMAT statement that applies only to TPL REPORT will be ignored by TPL TABLES. The reverse is also true, so there is no problem with statements that apply to only one of the applications.

There is a small possibility that you could put a FORMAT statement in the system profile that is meaningful in both applications but has different results. Or, you might wish to set a default for one application that is inappropriate for the other. In the unlikely event that this occurs, we recommend one of the following. You can install TPL REPORT in a separate directory, or you can make a new copy of the profile in the directory where you are running your report job(s).

Installation Steps

The exact installation procedure depends upon the platform on which you are installing TPL. Specific directions can be found on the CD jewel case that comes with your software.

So that users do not have to start tpl using full paths or modify their **.profile** PATH statements, you may wish to use the **ln** command to link some TPL programs into directories that are already in their paths; e.g. **/usr/bin**. The programs that should be linked are:

tpl
rerun
codebook
encaps
psp
report

Detailed Description of Setup Prompts

When you begin the setup program, it displays some introductory information on your screen and begins asking questions about the installation. Always remember to press the **<Enter>** key following your response.

Where Do You Want the System Installed?

Prompt:

Please specify the full path of the directory which is to RECEIVE the TPL system:

Response:

You must specify the full path. Relative paths will be rejected.

Printer

Prompt:

Do you standardly use a PostScript printer?

Response:

If you will be working with a line printer, respond with **N** for no.

If you will be working with a PostScript printer, respond with **Y** for yes. If you choose PostScript as the default output option, the FORMAT statement

```
POSTSCRIPT = YES;
```

will be included in the system profile along with some default PostScript font specifications. You can change any of these defaults after installation by editing **profile.tpl**, or you can override them with FORMAT statements in your format requests.

Note that the POSTSCRIPT statement must be set to YES for correct printing of reports on a PostScript printer. Likewise, if you are using a line printer, your reports cannot be printed correctly if POSTSCRIPT is set to YES.

Page Size

TPL REPORT will automatically format your reports according to the page size you specify in answer to the next prompts. Your answers will depend on the type of printer, size of paper and the type style you wish to use.

For Line Printers (non-PostScript)

Prompt:

Please specify the default page width for your printer.

The most common values are 80, 96, 132, and 160.

Response:

For 8 1/2 by 11 inch paper, the usual response is **80**.

Prompt:

Please specify the default page length for your printer.

The most common values are 66 and 88.

Response:

For 8 1/2 by 11 inch paper, the usual response is **66.**

For PostScript Printers

You can pick among several standard page types, or you can specify page size in inches, centimeters, points or characters. Fractions should be expressed as decimal numbers. For example, a page width of 8 1/2 inches should be entered as 8.5 inches.

When using PostScript, it is best to express page size in something other than characters. This is because, with PostScript, you can choose different character sizes. If page size is expressed in characters, the size of the page will vary as the character size changes. This result is usually undesirable.

Editor

TPL REPORT has been designed so that you can use the text editor of your choice to create codebooks, report requests, and format requests. Any editor that creates standalone (ASCII text) files is acceptable. In the next part of the installation process, you will link TPL REPORT to your editor.

Prompt:

If a TPL job fails because of a request error, the job will be put into the selected editor. When editing is completed and the editor terminated, TPL processing will resume. The default editor is the UNIX editor, **vi**.

Please type

<ENTER>	if you wish to use the currently selected or default editor,
none <ENTER>	if you do not wish to use an editor.
editor-name <ENTER>	if you wish to select an editor.

Response:

If you have an editor other than **vi** on your system, you may wish to enter its name. However do not use a word processor which inserts formatting information into your file unless there is an option to save the file in "text only" mode.

If You Change Your Mind

You are now given the option to change any of your answers to the questions you have been asked. Even if you respond with no to this prompt you will still be able to change the effect of your responses by editing **profile.tpl** after installation is complete.

If You Have Multiple Printers Connected to Your Computer

TPL REPORT will direct its output to the default printer for your computer. If you wish to change this, you may modify the profile statement

```
Print  Command = 'lp';
```

For example you might replace the command with

```
Print Command = "lp -dpost";
```

where **post** is the name of your PostScript printer. Note that if different people wish to use different printers they should create local profiles with different print commands.

APPENDIX E

Run Instructions (UNIX/Linux)

INSTRUCTIONS FOR RUNNING TPL REPORT UNDER UNIX

General Information

Editor

TPL REPORT is designed to allow you to use **vi** or another editor of your choice to create codebooks, report requests and format requests. Any editor that creates standalone UNIX files is acceptable.

If you have installed TPL REPORT so that it can access your editor and you are running a job that stops because of errors, TPL REPORT will prompt you to find out if you want to transfer to the editor. If you are transferred to the editor, TPL REPORT will automatically resume processing when you are finished with your editing.

Where to Run Jobs: Paths and Files

We do not recommend that you mix your files with system files by putting your own TPL-related files in the TPL system directory. Instead, put your own files in one or more other directories and run your jobs from those directories. This will work best if you put the path to TPL REPORT or TPL TABLES in your path command in **.profile** though it is not required. If you do not, you can start a TPL REPORT job by including the path information in your command.

It is a good idea to run your TPL REPORT jobs in the directory where your TPL-related files are stored, because then you can simply provide the file names without including path information. For any of your files that are not in the directory where you are running a job, you may include the path information.

How to Stop

The easiest way to stop a TPL job in the middle of processing is to type:

```
<Ctrl>C
```

If this doesn't work, open a new window and type:

```
ps -A
```

Then type:

```
kill -9 pid
```

where **pid** is the process id associated with the TPL process.

Note on Running in Background

All processes can be run in background, with the exception of **rmtpl.** The prompt for background processing and the **-b** argument are described under **How to Run a Report Request**.

Codebook Processing

Prepare your codebook (data description) file using your editor. We recommend that you save it with the same name you use at the beginning of the codebook. For example, if you name the codebook **survey** with the codebook statement **begin survey codebook**, save your codebook file as **survey.cbk**. The codebook file you have prepared will be referred to as the codebook source.

Note If you have a partial codebook source that needs to be completed with information from the data or if your data has changed such that new condition values need to be added for control variables, run TPL **conditions** first to create a complete or updated codebook source.

How to Run *codebook*

To run the codebook processor, type:

```
tpl  codebook  <Enter>
```

The codebook processor will display the prompt:

```
Please type the name of your codebook request  and <Enter>

==>
```

If you have a codebook source named **survey.cbk**, as in the example above, you would type:

```
survey.cbk  <Enter>
```

Codebook Command Line Arguments

You can bypass the prompt for the codebook source name by entering your codebook command as:

```
tpl  codebook  -c  cbsource  <Enter>
```

where **cbsource** is the name of your codebook source.

Error Handling

As the codebook processor runs, it will display your codebook on the screen along with messages about any errors it finds. All information displayed on the screen during processing will be stored with the same name as the codebook except that it will be capitalized and **.O** will be appended to the name. If your codebook is named **survey**, the processing information will be stored in a file called **SURVEY.O.**

If the codebook processor finds errors in your codebook, you will need to correct them with your editor and process the codebook again. If any syntax errors are found in the codebook, processing will stop. For most other types of errors, processing will continue to the end of the codebook. In that case, you will probably want to look for the error messages in the file containing processing information (e.g. **SURVEY.O***).*

When your codebook has been processed successfully with no errors, the **.O** file will be deleted and the **.L** codebook abstract file will take its place.

Codebook Abstract

The codebook abstract name ends with **.L** (e.g. **SURVEY.L**). The abstract includes the name of the codebook source file, the date and time of processing, and the TPL version number. In addition, it contains a list of the codebook variables in alphabetical order along with each variable's size and location within a record. This information is particularly useful if you have an alignment problem between your codebook and your data file. You may also find the abstract useful as a quick reference when preparing your report specifications. If you are creating a codebook describing a **CSV** or other type of delimited file or a database, the information in the abstract will differ slightly.

Producing A Codebook Source with the *conditions* Procedure

If you do not already have a codebook source, TPL **conditions** can be used to create a full codebook source from a partial one. It can also be used to update a codebook source if the data has changed such that new condition values need to be added for control variables.

Prepare your partial codebook with your editor as described in the Appendix called "TPL Conditions".

How to Run a *conditions* Request

To run a **conditions** job, type:

```
tpl conditions <enter>
```

The program will prompt you for your partial codebook source (with missing conditions).

It will then prompt you for your data file or database name. If the program cannot find the name it will ask whether the name is a SQL database name. Answer **y** or **n** as appropriate. If you answer **n**, you will be re-prompted for the data file name.

Finally you will be asked for the name of the completed codebook source you wish to create. You can use the same file name for your original source and your completed source. If you do, the completed source will be placed on top of the original source. The original source will be saved, along with a few extra statements, in the **.O** output file until the completed source has been created successfully. Thus, if there are any errors or problems that interrupt the creation of the completed source, you do not risk losing your original source. It will still be available in the **.O** file.

You will then be asked if you want to run the job in background.

If your codebook describes a database, you will be prompted for database user name, password, database server, etc.

The resulting complete codebook source file can be passed directly into a TPL codebook run or it can be edited to improve condition labels before codebook processing.

Command Line arguments for *conditions*

-c incomplete-codebook-source
-s complete-codebook-source
-d data-file (if your data is fixed format or delimited, e.g. csv)
-q database (if your data is in an SQL database)
-U user (SQL only)
-S database-server (SQL only)
-P database-password (SQL only - password may need quotes)
-b to run job in background

Error Handling

During the first part of TPL **conditions**, error handling is identical to codebook error handling as described above. After the original, incomplete codebook has been found to be valid, the program moves to the data reading step to get the information it needs to complete the codebook. Data errors such as incorrect characters in observation fields are added to the **.O** file. Data errors will not stop processing and will not put you into an editor.

Producing Reports with the *report* Procedure

Prepare your TPL report request with your editor. In the **USE** statement at the beginning of a report request, you can refer to the codebook using the same name you used in the **begin codebookname codebook** statement. Using the name **survey** shown in the example above, you would say **use survey codebook;** at the beginning of your TPL report request. TPL REPORT will know to look for a codebook object file called **SURVEY.K** for descriptive information about your data file. Path names are allowed in the USE statement.

Store your report request with any valid UNIX file name, for example, **survey.req**. You may also have an optional format request giving detailed specifications for formatting your reports. The format request can have any valid UNIX file name, for example, **survey.fmt**.

How to Run a Report Request

To run TPL REPORT, type

```
tpl  report  <Enter>
```

TPL REPORT will display the prompt

```
Please type the name of your TPL Report request and <Enter>:

==>
```

Using the name from the example above, you would type:

```
survey.req   <Enter>
```

TPL REPORT will display the prompt

```
Please type the name of your data file and <Enter>

==>
```

Your data file can have any valid UNIX file name. To continue the "survey" example, we will assume that your data is called **survey.dat**. You would type:

```
survey.dat <Enter>
```

If you are running against a database rather than a file, you should enter the database name. If you have entered a database name or an incorrect file name you will be asked if the name is a SQL database name. If it is, answer **y** and processing will continue with questions about your database user name, password, and server. If you answer **n** and you will be re-prompted for your data file.

TPL REPORT will display the prompt

```
Please type the name of your format request and <Enter>
or just type <Enter> if you do not wish to provide a format
request file:

==>
```

Often you will not have a format request. In this case, simply press the **<Enter>** key to continue. Otherwise, type the name of your format request. For example:

```
survey.fmt   <Enter>

Do you wish to run this request in background?

y or n ==>
```

Answering **y** to this prompt is the proper way to run TPL REPORT as a background process. Don't just use **&**. When the job is put in background, all output except the reports goes to the **output** file. Nothing is displayed on the screen and you are not put into your editor when errors are found in your request.

```
Do you wish to be notified when the request completes?

y or n =>
```

If you answer **y** to this prompt, when the job completes a message will appear on the screen telling whether the job has completed successfully or whether errors were detected in the request. In any case you should examine the **output** file in the TPLR subdirectory. The TPLR subdirectory is explained later.

Report Command Line Arguments

If you wish, you can bypass some or all of the prompts by entering your **report** command with any of the following parameters. Note that **-e**, **-E**, **-N**, and **-O** will be explained more fully later

-r requestfile	where requestfile is the name of your report request file
-f formatfile	where formatfile is the name of your format request file
-d datafile	where datafile is the name of your data file
-b	to run job in background
-n	to notify when job has completed
-E	to request only a partial display of output on the screen when running in foreground. For details, see the section on controlling screen display.
-e	if PostScript is set, convert reports into Encapsulated PostScript.
-V	if PostScript is set, convert reports to CSV (delimited) format.
-N nnnn	use TPLRnnnn as TPLR subdirectory where nnnn is a user selected number of one to four digits. If there is already a directory of that name, create a new number.
-O nnnn	use TPLRnnnn as a new TPL REPORT subdirectory overwriting any existing subdirectory of that name.
-i includepath	where includepath is the path to the directory where %include files are located. Use if you have include files in a directory other than the run directory. For details, see the section "Path for INCLUDE files".

-U database-user-name (SQL only)
-P database-password (SQL only - password may need quotes)
-S database-server (SQL only)
-q database-base (SQL only)

Example tpl report -r survey.req -d survey.dat <Enter>

Report Request Processing

As TPL REPORT processes your request, it will display the request on the screen along with messages about any errors and other information to show you the status of the job. If there are any errors in the report or format requests, you will be asked:

If you wish to edit your request and continue

respond with 'y' to the prompt. You will then be put in your editor. Upon termination of your editing session you will be returned to TPL and processing will continue. A response of 'n' will terminate the TPL session

If you answer **y**, you will be allowed to correct your errors and processing will continue. If you can't figure out your errors from what is displayed on the screen, you should answer **n** to the prompt and examine the error messages in your **output** file (described later). When you have fixed your errors you should start your report request again. Processing will stop immediately if a syntax error is encountered. For most other errors, processing will continue to the end of the request.

If no request or format errors are found, TPL REPORT will draw an hour glass on the screen as it begins to read your data. You will be able to tell how much of your data has been processed by the amount of sand that has fallen to the bottom of the hour glass. If your codebook does not match your data or if there are errors in the data, messages will be displayed at the bottom of the screen.

When TPL REPORT has finished processing your data and calculating the values for your reports, it will tell you whether the job has completed successfully.

You may examine the **output** file in the TPLR subdirectory to review any data errors and determine whether you should print your reports. The **output** and **report** files are described in the next section.

Example

```
tpl report -r survey.req -d survey.dat -b -n <Enter>
```

Since **-b** and **-n** have been specified, no output will be displayed on the screen except for the final status of the job. You will not be given the opportunity of correcting errors and continuing processing. Instead you must examine the output file in your TPLR subdirectory and resubmit your job if an error is found. If the job has run correctly, you may print your reports.

Controlling the Amount of Screen Display in Foreground

You can use the statement **display output = no;** in your profile to reduce the amount of screen display when running in foreground.

You can also use the **-E** command line option with both codebook and report runs. It provides a convenient way of running jobs in foreground, because it lets you see what is happening but reduces the volume of screen display. Display of codebooks or requests is suppressed. If an error is encountered in your codebook or request, the output ends with the error message and the preceding line of your codebook or request. This way, you can often see where the error is without looking at the

entire output file. Note that this option will not work if you have the statement **display output = no;** in your profile.

The TPLR Subdirectory

Each time you run TPL REPORT, it creates a subdirectory to hold the files it needs to create your reports. The subdirectory always has the name **TPLRnnnn**. The process id is used for the **nnnn** part of the subdirectory name, unless there is already a subdirectory using that number. You can find these subdirectories with the UNIX command **ls TPLR***. If you do not wish to let TPL REPORT select your TPLR subdirectory number, you can specify one yourself by using **-O nnnn** or **-N nnnn** on your command line. If you use **-N**, TPL REPORT will use **nnnn** only if there is not already a subdirectory in your current directory named **TPLRnnnn**. If such a subdirectory already exists, the **-N** argument will be ignored and a new numbered subdirectory will be generated. If **-O nnnn** is chosen, the subdirectory will be **TPLRnnnn** regardless of whether there was already one by that name. The old one will just be overwritten.

Most of the files that go into a subdirectory are illegible from your point of view and are erased when they are no longer needed to make your reports. However, there are two types of files in the subdirectory that you will want to see.

The File Called output

The first of these files is called **output.** It contains all of the information that was displayed on the screen while your job was running — all except the reports, that is.

If the messages go by on the screen too fast for you to read while your job is running, you can find them in the **output** file. If you run your job in the background or leave your computer while the job is running, you can find all the information that was displayed on the screen in the **output** file.

To help you keep track of your jobs, the **output** file contains the names of your data and request files, the date and time of execution for each part of the job, the TPL version number, and, at the end, the name of the TPLR subdirectory in which it was created.

The Report Files

The other files that you will want to see are the files containing the completed reports. Multiple REPORT statements can be included in the same job. A separate report file is created *for each* REPORT statement. A report file name begins with the **report-name** assigned in the REPORT statement and ends with the suffix

.rep . For example, if the report-name is IND_REPT, the report is saved in a file called **ind_rept.rep**.

If **Postscript = yes**; is specified, the suffix for the report files is **.ps**, for example **ind_rept.ps** .

Printing reports and output

When a TPL REPORT job ends you will be asked:

```
Do you wish to print your TPL output file?

y or n ==>
```

and

```
Do you wish to print your reports?

y or n ==>
```

TPL REPORT will print your output and/or report files if you respond with a **y** to the appropriate prompts. If you wish to prevent these prompts you may add **print output = yes;** (or **no**) and/or **print reports = yes;** (or **no**) statements to your **profile.tpl** or format request. See the FORMAT chapter for explanations of these commands. They are especially useful if you are running your job in background. If you want to print the report or output files later, you can use the standard UNIX **lp** command. You can also review the report and output files on the screen with the UNIX commands **more** report-name**.rep** and **more output**. or you can look at either file with your text editor.

Please note that the report files are formatted for the printer rather than the screen. If you display them on the screen and you have included horizontal lines in your reports, you may notice that some of the lines are missing or out of place. Also, you should choose a fixed-width (non-proportional) font for screen display. Otherwise, the columns will not be aligned properly. Regardless, the reports will print correctly.

PostScript Reports

If you are in PostScript mode, the file report-name**.ps** is a PostScript file. It should not be sent through a PostScript filter before printing. It also does not look like a report when listed with **more** or examined in your editor.

If you have specified a value for **DISPLAY NAME** in your profile, you will be asked if you wish to display your report in the PostScript displayer you have speci-

fied. If you answer **yes**, the displayer will open your report in a separate process. If you have multiple reports, each will be opened in a separate process.

The **output** file is never a PostScript file. It can be read using **more** or your editor. If you wish to print it on a PostScript printer, you may do so by a command such as **psp TPLR123/output | lp** . See the Appendix called **Utilities** for additional information on the **psp** program.

EPS and CSV Exports

If you have run your request in PostScript mode, you will get two additional questions:

Do you wish to create Encapsulated PostScript files for use by other programs?

and

Do you wish to create delimited (CSV) files for use by other programs?

Use of **-e** on the command line will cause **EPS** files to be created without the **eps** prompt appearing. Use of **-V** will cause the **CSV** files to be created without the **csv** prompt appearing. See explanations of these export formats below.

Another way to contol export prompts when running jobs in foreground is to put either of the following in your **profile.tpl** or format request. For each, the *value* should be **yes**, **no**, or **prompt**.

EPS OUTPUT = *value;*
CSV OUTPUT = *value*;

Encapsulated PostScript (eps)

Many desktop and professional publishing systems allow importation of PostScript files provided they are in **EPS** format. If you answer **y** to the Encapsulated PostScript prompt, TPL REPORT will convert your report-name**.ps** files into a collection of EPS files. If you answer **n**, you can create EPS files later by moving into the TPL REPORT subdirectory and typing:

encaps . 0 report-name.ps <Enter>

ENCAPS will report the names of the **EPS** files as they are created.

Other options are available, such as naming the directory for the output instead of specifying '.' for the current directory and running silently with no reporting (1 instead of 0). The current options will be displayed on the screen if you type:

encaps <Enter>

The **ENCAPS** program works only with **.ps** files created by TPL software. It cannot be used with PostScript files created by other programs.

Reports can be most conveniently imported into another system if each page is in a separate file. Consequently TPL REPORT creates one file for each page of report output. The files are named by page and report number. For example, if you have a two page report followed by a one page report, the report output will be divided into three files with the following names:

```
P1R1.eps
P2R1.eps
P1R2.eps
```

The ***report-name*.ps** files containing the complete report outputs will still be available for printing or you can print individual pages of your reports by printing the EPS files.

If you have more than one report on a page, they will all be contained in the same **EPS** file.

Delimited or Comma Separated Variable (CSV) Files

Comma Separated Variable (CSV) format is a common data interchange format. TPL can read **CSV** and other types of delimited files as data and can output reports in CSV format for use by other programs.

Each report with data produces a separate **CSV** file in your **TPLRnnnn** directory. The reports are labeled ***report-name1*.csv, *report-name2*.csv,** etc where ***report-name1*** is the name of the first report, etc.

Path for INCLUDE files

For report or codebook runs, if you have %INCLUDE files that are in a directory other than the run directory, you can use the **-i** argument to enter the path to the directory where the %INCLUDE files are located.

For example, if you have an include file called **recodes.txt** that is located in the directory called **/usr3/tplwkgrp/ALB.FILES**, you can use the **-i** argument on the command line as follows:

```
-i  /usr3/tplwkgrp/ALB.FILES
```

Then in your %include statement, use the file name:

```
%include recodes.txt
```

You may only have one include path.

Another way to access include files in another directory is to use the UNIX **ln** command to make the include files appear to be in the local directory.

Removing Subdirectories with the *rmtpl* Command

The **rmtpl** command makes it easy for you to erase TPLR subdirectories that you no longer want to keep.

How to Run *rmtpl*

To erase a subdirectory, first be sure that you are in the directory that contains the subdirectory. Then type the command

```
rmtpl  nnnn  <Enter>
```

where **nnnn** is the number of the subdirectory you want to erase.

You can delete multiple TPLR subdirectories by including multiple numbers on the command line. For example,

```
rmtpl  123  456  7834  <Enter>
```

To delete all TPLR subdirectories contained in the current directory, type:

```
rmtpl  all  <Enter>
```

Note If you also have TPL subdirectories created by TPL TABLES in the same directory, the command **rmtpl all** will remove these subdirectories as well.

Creating Your Own Environment with the profile.tpl File

The TPL REPORT installation process creates a file called **profile.tpl** and puts it in the TPL REPORT system directory. This file allows TPL REPORT to adjust to your operating environment.

The **profile.tpl** file contains statements that you can change with your editor after installation if something changes in your operating environment. For example, if you begin using a PostScript printer, you might want to edit **profile.tpl**.

You can also change report format defaults by including FORMAT statements in **profile.tpl**. For example, if you always want your reports left-adjusted on the page, you can make it a default by including the FORMAT statement **align reports left;** in **profile.tpl**.

If you want to leave the system profile unchanged, but use a different profile for a particular set of jobs, you can make a copy of **profile.tpl** in the directory where you are working and change that copy to fit the reports you are preparing. The profile in the directory where you are working will override the one in the TPL REPORT system directory.

If your copy of TPL REPORT is being shared over a network, you may wish make a copy of **profile.tpl** that is appropriate for the way you want to use TPL REPORT.

Note to TPL TABLES Users

If you have installed TPL TABLES and TPL REPORT in the same directory, the two applications will be sharing the same **profile.tpl**. In this case, you should be aware of the fact that FORMAT statements entered in the profile will be shared by the two applications. In addition, the words TABLE and REPORT are equivalent. For example, the FORMAT statement ALIGN TABLES LEFT; will align reports to the left as well.

Any FORMAT statement that applies only to TPL REPORT will be ignored by TPL TABLES. The reverse is also true, so there is no problem with statements that apply to only one of the applications.

There is a small possibility that you could put a FORMAT statement in the system profile that is meaningful in both applications but has different results. Or, you might wish to set a default for one application that is inappropriate for the other. In the unlikely event that this occurs, we recommend one of the following. You can install TPL REPORT in a separate directory, or you can make a new copy of the profile in the directory where you are running your report job(s).

Piping Data to TPL REPORT

TPL REPORT supports standard piping of data into a request and also supports the more flexible named pipes.

Standard Piping

Standard piping is done by using just the standard '|' symbol plus the TPL REPORT keyword **%pipe.**

An example is:

```
cat datafile | tpl report -r request -d %pipe
```

The piped input may of course come from the output of any program which writes to the standard output (console). The hourglass is not shown while data is being read.

With this type of piping, TPL REPORT reads the piped data as if it were coming from the standard input (the keyboard). Thus, the following rules apply:

1. Both the **-r** and **-d** arguments must be included and be correct or the job will fail to execute.
2. TPL REPORT will not prompt you for missing or incorrect arguments. Since the standard input (keyboard) is used for the pipe, there is no way to respond to prompts using the keyboard.
3. Jobs can only be run in foreground. You cannot use the **-b** argument to run TPL REPORT in background.

Named Pipes

Named pipes or FIFOs provide a more flexible method for connecting the output of one program to the input of another. TPL REPORT treats a named pipe just like a file except that the hourglass is not displayed when a named pipe is used.

Named pipes are usually preferable to the type of piping described above as "standard piping". Since the named pipe is not the standard input, but rather a separate entity with its own name, the keyboard is free for reponding to prompts. In addition, you can use the **-b** argument to run jobs in background.

To use named pipes, first create a named pipe using the **mknod** command:

```
mknod /dev/your-name p
```

where ***your-name*** is whatever you want. The pipe need not be created in ***/dev*** though this is customary. The **p** is required to indicate that the node is to be a pipe. The pipe need only be created once as it will stay around between jobs.

Now you can direct the output from your data-generating program into the pipe. Start TPL REPORT with the pipe name as the input file. TPL REPORT detects that the input file is a pipe rather than a regular file and modifies the processing as appropriate.

Suppose your pipe name is **/dev/my_pipe**. You can pipe a data file called my_data into TPL REPORT with the following sequence:

```
cat my_data > /dev/my_pipe &
tpl report -d /dev/my_pipe
```

Most UNIX programs which write output to a user-specified file can write their output to a named pipe and hence can pipe their output into TPL REPORT.

Silent Use of Pipes

Named pipes can be used to run jobs silently in background in such a way that there is no output on the screen. The following example shows how TPL REPORT can read data from a pipe and run without displaying even a process id on the screen.

```
cat datafile > named_pipe &
tpl report -r request -d named_pipe -b > /dev/null
```

Although TPL REPORT will run silently in this example, we will get a process id displayed from the **cat** program. In a real case, we would not be using **cat** to fill the pipe so there would be no problem.

For example, we can replace the **cat** with a trivial program called **pipe_fill** as follows:

```
main()
 {
  system("cat datafile > named_pipe &");
 }
```

Then pipe_fill will not display the process id so the following sequence will be completely silent:

```
pipe_fill
tpl  report -r request -d named_pipe -b > /dev/null
```

Common Error Messages

Error messages are intended to be self-explanatory. However, three common errors deserve special note.

Common Error Message 1

```
*** ERROR: A syntax error was discovered while processing
        'element'.  Look for the error at or before that point.
```

This message appears whenever there is a syntax error in a codebook, report request, format request or profile. Examples of syntax errors are misspelled keywords or punctuation errors such as a missing colon (:) or semicolon (;). The point at which TPL REPORT discovered the error is indicated by the element in quotes.

Example Following is an example showing the beginning of a REPORT statement and the error message that would result:

```
REPORT ONE 'Employee Data'
        AGE THEN SEX THEN SALARY;

*** ERROR: A syntax error was discovered while processing 'AGE'.
        Look for the error at or before that point.
```

Since the error was found when the variable name AGE was encountered, we can assume that there is something wrong with the name, or that an error preceded it so that it appears to be in the wrong place. In this example, a colon (**:**) is missing following the report title. TPL REPORT is looking for the colon when it finds the variable name AGE.

Common Error Message 2

```
*** ERROR: The variable 'variable name' is undefined.
```

A frequent cause of this error is a misspelled name. Another cause is a reference to a variable that has not yet been defined. For example, if a variable is created in a COMPUTE statement and used in a REPORT statement that precedes the COMPUTE statement, the computed variable is unknown to TPL REPORT when it finds it in the REPORT statement.

Example Misspelling of the variable name SALARY as SALRY produces the message shown below.

```
COMPUTE TOTAL_INCOME = SALRY + INTEREST;

*** ERROR: The variable 'SALRY' is undefined.
```

APPENDIX F

TPL Conditions (Unix Only)

WHAT IS TPL CONDITIONS?

The **tpl conditions** program converts partial codebook sources into complete codebook sources. In doing so, it saves you work in creating codebooks and also assures that the codebook source accurately describes the data. It works with codebooks for databases, fixed format files and delimited files such as CSV. The program can also be used to update a codebook source when the data file or database has changed in such a way that additional condition values are needed. **tpl conditions** fills in condition values and labels for all types of codebooks. For delimited (CSV) and database codebooks it also fills in field sizes. For database codebooks, **tpl conditions** fills in data types of observation variables such as **float**. See *Producing A Codebook Source with the conditions Procedure* in **Run Instructions (UNIX)** for details on how to run a **tpl conditions** job.

Control Variable Conditions

As the name of the procedure implies, the biggest use of **tpl conditions** is to fill in condition values for control variables. If the codebook is new, the condition value lists are presumably empty. In this case **tpl conditions** inserts all of the conditions found in the data for each control variable. The conditions are assigned default labels.

If the codebook is old and is merely being updated, all existing conditions and their labels are retained. **tpl conditions** just adds the new conditions found in the data. Where the new conditions are added depends upon the **display as** clause. If there is no **display as** clause or **display as sorted** is specified, the old and new conditions for a variable are intermixed and placed in sort order based on the value. If **display as listed** is specified, the old conditions are retained at the start of the condition list and the new conditions are placed at the end of the condition list in the order they are encountered.

When **tpl conditions** has finished, you may edit the new codebook source to provide better labels for the new conditions and to rearrange them if desired.

Note **tpl conditions** cannot update codebooks that contain groups.

Fixed Format Sequential File Example

The following is an incomplete fixed format sequential file codebook before **tpl conditions** has been run. Note that all fields must have a width since this is the only way TPL can identify the boundaries of a field. **aip** has no conditions but it must have parentheses. **Complainant**, **shift_** and **squad** all have some condition values.

```
Begin dispatch codebook ascii

dispatches 'Dis'-'patches' record level 0
   filler 6
   A_I_P 'A-I-P' control 1
    (
    )
   filler 5
   STREET1 '1STREET' char 4
   STREET2 '2STREET' char 4
   COMPLAINANT 'Complainant' control 30
    (
     'Alarm Panel'  = 'ALARM PANEL'
     'Blairs Florists/John'  = 'BLAIRS FLORISTS/JOHN'
     'Cowden,George'  = 'COWDEN,GEORGE'
     'Marion High School'  = 'MARION HIGH SCHOOL'
    )
   td obs 4
   tr obs 4
   ta obs 4
   tc obs 4
   unit 'Unit' char 4
   filler 36
   full_date 'Date' char 6
   shift_ 'Shift' control 1 display as sorted
    (
     'first shift'  = '1'
     'third shift'  = '3'
    )
   filler  1
   squad 'Squad' control 1 Display as listed
    (
     'unknown'  = ' '
     'squad 9'  = '9'
    )
End dispatch codebook
```

The following is the completed codebook source after **tpl conditions** has been run. Note that **filler** has been removed. Instead, the field following the filler has a start position. **Complainant** does not have a **display as** clause so the old conditions are sorted into value order along with the new values. **shift_** has a **display as sorted** clause so the old conditions are also sorted into value order with the new conditions. **squad** uses **display as listed** so the old conditions retain their order and the new conditions are added after them.

```
Begin DISPATCH codebook ascii
DISPATCHES "Dis" - "patches" Record Level 0
A_I_P "A-I-P" start 6 Con 1
 (
   = "A"
   = "I"
   = "P"
 )
STREET1 "1STREET" start 12 Char 4
STREET2 "2STREET" Char 4
COMPLAINANT "Complainant" Con 30
 (
   = " "
   = "7 AV STD"
   "Alarm Panel" = "ALARM PANEL"
   = "ALEXANDER,DICK"
   = "ARP,MICHAEL"
   = "BEETS,GENEVA"
   = "BEHNKE,MRS"
   "Blairs Florists/John" = "BLAIRS FLORISTS/JOHN"
   = "COOK,TOM"
   = "COOPER,DEB"
   "Cowden,George" = "COWDEN,GEORGE"
   = "CR 727"
   = "MARION 76/RANDI"
   = "MARION FIRE"
   "Marion High School" = "MARION HIGH SCHOOL"
   = "MATTESON,KENNETH"
   = "WORTMAN,DAVID"
   = "YATES,DOUG"
   = "YEISLEY,BILL"
   = "YIRKOUSKY,DARREL"
   = "YOUNG,MARVIN"
 )
TD "TD" Obs 4
TR "TR" Obs 4
TA "TA" Obs 4
TC "TC" Obs 4
UNIT "Unit" Char 4
FULL_DATE "Date" start 106 Char 6
```

```
SHIFT_ "Shift" Con 1
Display as sorted
 (
   "first shift" = "1"
   = "2"
   "third shift" = "3"
 )
SQUAD "Squad" start 114 Con 1
Display as listed
 (
   "unknown" = " "
   "squad 9" = "9"
   = "1"
   = "R"
 )

End DISPATCH codebook
```

Delimited (CSV) Sequential File Example

The following is a small incomplete CSV codebook before **tpl conditions** has been run. Note that sizes are not specified but field number is. Some of the fields have been skipped. For the field **complaint** some of the condition values have been provided. **aip** has no fields provided but it does have the required parentheses.

```
Begin dispatch_csv Codebook CSV (Head = Yes Delimiter = COMMA)
dispatch_csv Record
ID "id" Field = 1 Char
AIP "aip" Field = 2 Control ()
COMPLAINANT "complainant" Field = 6 Control Right Blank Fill
 (
   "ALARM PANEL"  =  "ALARM PANEL"
   "BLAIRS FLORISTS/JOHN"  =  "BLAIRS FLORISTS/JOHN"
   "COWDEN,GEORGE"  =  "COWDEN,GEORGE"
   "MARION HIGH SCHOOL"  =  "MARION HIGH SCHOOL"
 )
SQUAD "squad" Field = 22 obs
End dispatch_csv
```

The following shows the complete codebook after **tpl conditions** has been run. Field widths have been filled in as have condition values for **aip**. Conditions have also been filled in for complainant, Since there is no **display as** clause, the old conditions are sorted in with the new conditions.

```
Begin DISPATCH_CSV codebook CSV
(Delimiter = Comma  Head = Yes  Quote = '"')
```

```
DISPATCH_CSV "DISPATCH CSV" Record Level 0
ID "id" Report Error = No
Field = 1 char 5
AIP "aip" Field = 2 Con 1
 (
   = "A"
   = "I"
   = "P"
 )
COMPLAINANT "complainant" Field = 6 Con Right Blank Fill 28
 (
   = " "
   = "7 AV STD"
   "ALARM PANEL" = "ALARM PANEL"
   = "ALEXANDER,DICK"
   = "ARP,MICHAEL"
   = "BEETS,GENEVA"
   = "BEHNKE,MRS"
   "BLAIRS FLORISTS/JOHN" = "BLAIRS FLORISTS/JOHN"
   = "COOK,TOM"
   = "COOPER,DEB"
   "COWDEN,GEORGE" = "COWDEN,GEORGE"
   = "CR 727"
   = "MARION 76/RANDI"
   = "MARION FIRE"
   "MARION HIGH SCHOOL" = "MARION HIGH SCHOOL"
   = "MATTESON,KENNETH"
   = "WORTMAN,DAVID"
   = "YATES,DOUG"
   = "YEISLEY,BILL"
   = "YIRKOUSKY,DARREL"
   = "YOUNG,MARVIN"
 )
SQUAD "squad" Field = 22 Obs 1

End DISPATCH_CSV codebook
```

Error Detection In addition to producing a new codebook source, **tpl conditions** detects errors. For this example, the error messages were placed in **DISPATCH_CSV.O**. The field **squad** is described as **obs** but it has some letters in it. The following is the last part of the file **DISPATCH_CSV.O** where the errors are reported.

```
For record 255 Variable SQUAD:  'R' cannot appear in an ascii observation value.
For record 255 Variable SQUAD:  An observation value must contain a digit.
For record 256 Variable SQUAD:  'R' cannot appear in an ascii observation value.
For record 256 Variable SQUAD:  An observation value must contain a digit.
For record 267 Variable SQUAD:  'K' cannot appear in an ascii observation value.
For record 267 Variable SQUAD:  An observation value must contain a digit.
```

```
For record 268 Variable SQUAD:  'K' cannot appear in an ascii observation value.
For record 268 Variable SQUAD:  An observation value must contain a digit.
For record 280 Variable SQUAD:  'R' cannot appear in an ascii observation value.
For record 280 Variable SQUAD:  An observation value must contain a digit.

285 records read.
60 data errors were found.

End CODEBOOK CONDITIONS processing
```

SQL Database Example

The following is a small incomplete SQL codebook before **tpl conditions** has been run. Note that field sizes are not specified. Data types, such as **float**, have not been filled in for observation variables and no conditions are provided for the control variables. Instead, **get conditions from data** or **get conditions from *table*(*label,code*)** are used. Since this codebook describes a Sybase database with lowercase field names, each variable must have a **defines** clause.

```
begin sample codebook sql

employee defines "employee" table
company_id defines "company_id" obs
last_name defines "name" control from data
salary defines "salary" obs

company defines "company" table
company_name defines "name" control get conditions from data
company_id defines "company_id" obs
location defines "location" control
  get conditions from "locations"("location_name","location_id")
gross defines "gross" obs

company is parent of employee where company_id = company_id
```

After the incomplete codebook has been processed by **tpl conditions** the result is as listed below. **Last_name** and **Company_name** have conditions obtained from the data. **Location** has obtained its conditions from the **location_name** and **location_id** of the **locations** table. Field widths are filled in. Since the program was run against a Sybase data base, the **begin** statement references **Sybase** instead of **SQL**.The fields **Company_id** and **Gross** are now **obs float** instead of just **obs** and **salary** is now **obs money** and has a mask rather than just being **obs.**

```
Begin SAMPLE codebook Sybase
EMPLOYEE "EMPLOYEE" Defines "employee" table
COMPANY_ID "COMPANY ID" Defines "company_id" obs float 8
```

```
LAST_NAME "" Defines "name" control 9
 (
   "Balmer" = "Balmer"
   "Einstein" = "Einstein"
   "Gates" = "Gates"
   "Newton" = "Newton"
   "Watson" = "Watson"
   "Weeks" = "Weeks"
   "Weiss" = "Weiss"
 )
SALARY "SALARY" Mask Center $ 999.99 Defines "salary" obs money
COMPANY "COMPANY" Defines "company" table
COMPANY_ID "COMPANY ID" Defines "company_id" obs float 8
COMPANY_NAME "" Defines "name" control 12
 (
   "IBM" = "IBM"
   "Microsoft" = "Microsoft"
   "QQQ Software" = "QQQ Software"
 )
GROSS "GROSS" Defines "gross" obs float 8
LOCATION "" Defines "location" control 2
from "locations" ("location_name", "location_id")
 (
   "Arlington" = "01"
   "Everywhere" = "02"
   "Redmond" = "03"
   "New Carrollton" = "04"
   "Nowhere" = "05"
   "hometown" = "06"
   "Atlantis" = "07"
 )

COMPANY is parent of EMPLOYEE where COMPANY_ID = COMPANY_ID

End SAMPLE codebook
```

Comments

tpl conditions preserves comments in your codebook source. To assure accurate placement of your comments in the output, the comments should be put in one or more of the following places:

At the start of your codebook
Before the end codebook statement
Before a variable or record entry
Before a condition entry
Before an association statement

APPENDIX G

International

FORMATS, SYMBOLS AND LANGUAGES

Important

The CODEPAGE and COUNTRY statements described in this appendix are special statements that can be used in the profile for your jobs. If you add a CODEPAGE or COUNTRY statement to your profile, change a CODEPAGE or COUNTRY statement in your profile, or make changes to **country.tpl**, ***you need to restart TPL*** to activate the changes.

Your codebook must be processed with the same CODEPAGE and COUNTRY statements that you use when running your report requests. Otherwise, you will have conflicting standards. In particular, conflicts in CODEPAGE will cause the sort order to be scrambled.

Alphabets and Sort Order: The CODEPAGE Statement

The CODEPAGE determines the character set and sort order for your requests and tables. The default CODEPAGE will work with many languages. If you need additional characters for your alphabet, you can select a different CODEPAGE from those shown in the Appendix called "Character Sets". See also the CODEPAGE statement in the FORMAT chapter.

Entering characters, using them in labels and printing them. The most desirable way of entering characters is with a keyboard that is appropriate for the alphabet of the language you are using and an editor that supports it.

Any character that can be entered on the keyboard, either directly or by using **Alt** and the numeric keypad, can be used in TPL labels and other character strings such as condition values.

Characters not on your keyboard can also be entered by typing in their numeric code or by entering a character name.

Character Name. A character name is the name of a character preceded by **&** and terminated with **;.** For example **É** refers to the letter **E** with an acute accent above it. Character names are case sensitive. **é** is the letter **e** with an accute accent.The acceptable names are the names for the codepage you have selected. See *Special Character names* in the "Character Sets" Appendix. Use of character names instead of character codes has the advantage of being more portable. If you switch codepages, the table will look the same provided the character name is in both code pages. Also, if you are using a table for multiple purposes -- creating a pdf, creating a web page, and printing the table -- then use of a character name will in general result in a constant display of the character. Finally, table requests written using character names area easier to read than requests using character codes.

Character Code. A Character code is a **** followed by a 3 digit number which identifies the character. Three digits are always required. If the character can be represented by fewer than 3 digits, add leading zeros. For example, for a character represented by the code 65, enter **\065**.

The value **nnn** must be the ***decimal*** code for the character. Note that the character code tables in some software manuals show the octal or hexidecimal codes for the characters. If you are using this type of table, you must convert the code to its decimal equivalent. Character set tables showing decimal codes are included in the Appendix called "Character Sets".

In line printer (non-PostScript) mode, the characters will print correctly in tables if the corresponding characters are available on the printer. In PostScript mode, these characters will print correctly if they are included in the character set for the selected CODEPAGE.

Alphabet for user-specified names. If an alphabetic character is included in the character set for the selected CODEPAGE and the character can be entered on the keyboard, either directly or by using **Alt** and the numeric keypad, it can be used in names for variables, tables, and other items.

The Sort Sequence. The proper order for sorting depends on the character set used. TPL will use the sequence that goes with the character set selected by the CODEPAGE statement. The sort sequences for all character sets are stored in a file called **sort.tpl** that is installed in the TPL system directory.

Note Your should insert CODEPAGE *at the beginning* of your profile. You cannot do this until after TPL is installed.

The COUNTRY Statement

The COUNTRY statement is fully described in the FORMAT chapter of the manual. It lets you select standards for the characters to be used as decimal and thousands separators, the currency symbols and format, and formats for date and time. These standards are set in a file called **country.tpl** that is installed with TPL REPORT. US is the default country.

Note You should insert COUNTRY *at the beginning* of your profile. You cannot do this until after TPL REPORT is installed. Before inserting the COUNTRY statement, you should check to see if there are any decimal numbers already used in the profile. For example, decimal numbers can be used in the page size specifications. If you have any such instances, you should edit your profile to match your country standard.

Replacing Default English Text

If you regularly use TPL REPORT to produce reports in a language other than English, you may wish to replace the default text for labels such as NUMBER (**"Row"**) or title continuation (**" - Continued"**).

We recommend that you replace these labels by entering the appropriate FORMAT statements in your **profile.tpl** file. The new labels will then automatically apply to all of your reports.

APPENDIX H

Keywords

TPL REPORT KEYWORDS

The following words are TPL REPORT keywords. They should not be used as names for reports, variables, conditions, or codebooks.

ABS
AFTER
ALIGN
ALL
ALTERNATE
AND
AS
ASCENDING
ASCII
AT
AUTO
AUTOMATIC
BANK
BANKS
BEGIN
BINARY
BIT
BLANK
BLANKS
BOLD
BOTH
BOTTOM
BY
CELL
CELLFILE
CELLS
CENTER
CENTRE
CHANGE
CHAR
CHARACTER
CM
CODEBOOK
CODEPAGE
COLOR
COLOUR
COLUMN
COLUMNS
COMMAND
COMPRESS
COMPUTE
CON
CONDITION
CONDITIONS
CONT
CONTINUATION
CONTINUE
CONTINUED
CONTROL
COPY
COUNT
CREATED
CSV
DATA
DATE
DECIMAL
DEFAULT
DEFINE
DEFINES
DELETE
DELIMITER
DESCENDING
DISPLAY
DIV
DIVIDE
DIVIDER
DO
DOUBLE
DOWN
EACH
EIA
EJECT
EMPTY
END
EPS
EOF
EQUAL
EQUALS
EVALUATED
EVERY
EXCEPT
EXCLUDE
EXTRA
FETCH
FIELD
FILE
FILL
FILLER
FLOAT
FMEDIAN

FONT
FOOTNOTE
FOOTNOTES
FOR
FQUANTILE
FROM
GET
GRAND
GRANDTOTAL
GRAY
GREATER
GREY
GROUP
HEAD
HEADER
HEADERS
HEADING
HEADINGS
HEADNOTE
HEADS
HIERARCHIES
HTML
I
IF
IN
INCH
INCHES
INCOMPLETE
INCREMENT
INDENT
INPUT
INS
IS
ITALIC
JUSTIFIED
JUSTIFY
KEEP
KEY
LABEL
LABELS
LAST
LEADING
LEFT
LENGTH
LESS
LEVEL
LINE
LINES
LISTED
MARGIN
MARKER
MASK
MAX
MAXIMUM
MEAN
MEDIAN
MEMORY
MIN
MONEY
MONITOR
NAME
NAMES
NO
NORMAL
NOT
NOTE
NULL
NUMBER
NUMBERS
NUMERIC
OBS
OBSERVATION
ODBC
OF
ON
OR
ORACLE
OTHER
PAGE
PAPER
PARENT
PATH
PERCENT
PLAN
POINT
POINTS
POST
POSTCOMPUTE
POSTSCRIPT
PRIMARY*
PRINT
PT
PTS
QUANTILE
QUANTILES
QUOTE
RANK
RANKED
RECODE
RECORD
REDEFINES
REPEAT
REPEATS
REPLACE
REPORT
REPORTS
RETAIN
RIGHT
ROTATE
ROUND
ROW
ROWS
RULE
RULES
SCALE
SELECT
SEQUENCE
SET
SHADE
SHIFT
SIB
SIBLING
SIDE
SKIP
SORT
SORTED
SPACE
SPACES
SPAN
SPANNER
SPANNERS
SQL
SQRT
START
STARTS
STATCAN
STDERR
STDEV
STDEVP
STOP
STUB
STUBS
SUB
SUBSTR
SUBSTRING
SUBTOTAL
SUP
SUPER
SYBASE
SYM
SYMBOL
TABLE
TABLES
TABULATE
TEXT
THAN
THEN
TITLE
TITLES
TO
TOP
TOTAL
U
UNDERLINE
UNJUSTIFIED
UNJUSTIFY
UNLESS
UNSIGNED
UP
USE
USING
VALUE
VALUES
VAR
VARIABLE
VARIABLES
VARP
VARYING
WAFER
WAFERS
WEIGHTED
WEIGHTING
WHERE
WIDTH
WITH
YES

* Codebook only. You can continue to use this word as a variable name, if you precede it with a : in the codebook. For example, :PRIMARY

APPENDIX I

Limits

SUMMARY OF FEATURES AND SYSTEM CONSTRAINTS

Platforms and Operating Systems

Windows 98, XP, 2000, Vista.

UNIX platforms, including Sun and HP.

Can be ported to other UNIX platforms.

Contact QQQ Software for current list.

Minimum Hardware Configuration

Hard disk space: 30 megabytes

Printer: any

Optional Hardware

PostScript printer: On UNIX systems, a PostScript printer is required to print PostScript reports. On Windows systems, PostScript reports can be printed from TED on any printer. When PostScript reports are inserted in documents with desktop publishing software, a PostScript printer may be required to correctly print the reports. If you convert PostScript reports, or documents containing PostScript reports, to Adobe Acrobat PDF format, they can be printed from Adobe Acrobat Reader.

Hard disk space: The installed system occupies about 30 megabytes of hard disk space. Additional space is needed for temporary work files and for your data and output reports. Alternate drives can be substituted for anything other than the installed TPL REPORT system.

Features/Constraints

There are very few fixed limits in TPL REPORT. The available computer resources are allocated according to the unique requirements of each job so that space not needed for one feature can be used by another. Thus, it is highly unlikely that you will ever encounter a limitation on the size of your job. If you do, please contact Software Support for suggestions.

Maximum number of reports per request: no limit

Maximum number of variable references: no limit

Maximum number of values for a single control variable (including variables created by RECODE statements): no system limit, although performance may degrade with many hundreds of thousands of values, depending on the capacity of your computer and what you are doing with the variable.

Maximum columns per report: no known limit

Maximum print label length: no limit

Maximum record types and groups in codebook: 30

Input data file requirements:

Record formats: fixed length records with data fields in fixed columns; variable length CSV (comma separated) and other types of delimited files

Datafile type: sequential

Maximum record length: 32,764 bytes for fixed length records; 50,000 bytes for CSV and other delimited files

Datafile organizations: flat (single level) and hierarchical (multi-level)

Data field types: character (ASCII), binary and floating point (single or double precision)

SQL databases: The TPL-SQL Database Interface is optional. For Window systems, databases can be accessed via ODBC. For UNIX systems, contact QQQ Software for the current list of supported database systems.

Accuracy of computed results: Computations are done in ANSI standard double precision floating point with special code to prevent comparison errors introduced by radix conversion.

Format for codebooks, report requests and format requests: free format

Variable name format: up to 30 characters long, starting with letter, # or underbar(_), and containing only letters, digits, # and _

Statement types: report, sort, subtotal, grand total, select, recode, compute, conditional compute and use

APPENDIX J

Utilities

STAND-ALONE UTILITY PROGRAMS

Several stand-alone utility programs are installed with TPL REPORT. You may find some of these programs useful in applications other than TPL REPORT.

COMMENT

Location

Windows: Installed in the TPL system directory

UNIX: installed with TPL in the tpldebug subdirectory

File Name

comment.exe (PC version)
comment (UNIX version)

Purpose

Mainframe TPL uses @ to delimit comments. TPL for Windows or Unix uses /* and */ to delimit comments. If you are converting mainframe TPL codebooks to use with TPL for Windows or Unix, you can use the comment program to automate part of the process by converting the comments for you.

The comment program converts comments delimited by @ into comments delimited by with /* */ pairs.

Instructions

Execute the program from the command line. The required arguments are source file and target file. For example:

```
comment  survey.mf  survey.cbk  <enter>
```

where **survey.mf** is a mainframe TPL codebook. The codebook with converted comments will be stored in the file **survey.cbk**.

Note The TPL CONVERT system can be used to automate the entire conversion process for codebooks. If you are converting codebooks and do not already have the CONVERT system, contact QQQ Software for details.

FOR_WORD

Note FOR_WORD is a public domain program.

Location

Windows: Installed in the TPL system directory

UNIX: installed with TPL in the tpldebug subdirectory

File Name

```
for_word.exe  (if Windows)
for_word  (if UNIX)
```

Purpose

To take a file that was prepared with a line editor and convert it to word processing format. By "line editor", we mean a program that puts one or more return characters at the end of each line. By "word processor", we mean a program that works with paragraphs rather than lines. The FOR_WORD program will convert a line editor file for use with a word processor by removing the return characters from lines within paragraphs.

Instructions

The program is self-documenting. On the command line, type

```
for_word <Enter>
```

Instructions will be displayed on the screen.

HEXLIST

Location

Windows: Installed in the TPL system directory.

UNIX: installed with TPL in the tpldebug subdirectory

File Name

hexlist.exe (if Windows)
hexlist (if UNIX)

Purpose

The hexlist program displays the contents of a file as hexidecimal values and, when possible, as ascii characters. Where there is no ascii character equivalent for the hexidecimal value, a **%** symbol is displayed on the character line.

The hexlist program can be very useful in identifying problems in a data file when the file has errors or is not in the format that you expected.

Instructions

On the command line, type

```
hexlist arg1 arg2 arg3 <Enter>
```

where

arg1	is file name
arg2	(optional) is a line width <= 75. 75 is the default
arg3	(optional) indicates that the file should be opened in ascii rather than default binary mode. It must be a lower case letter “a” or the word “ascii” (not in quotes).

If you want the file opened in ascii mode, you must provide both arguments 2 and 3.

If the file is opened in binary mode (the default), all characters in the file, including any end-of-record or end-of-file indicators, will be displayed.

If the file is opened in ascii mode, carriage returns and end-of-file markers will not be displayed. Line feeds will be displayed.

On Windows systems, most ascii files have a carriage return and line feed at the end of each record and a control-Z at the end of file.

The hexidecimal codes for these end-of-record and end-of-file characters are:

```
0D    <CR>
0A    <LF>
1A    control-Z
```

UNIX Note If you are working with a UNIX system, the binary/ascii distinction is irrelevant since you will get the same result either way. Most UNIX ascii files have a line feed (hexidecimal 0A) at the end of each record.

How to Stop

If you have a large file, you may wish to stop the hexlist after displaying just a part of it. You can stop the hexlist by entering <Ctrl><Break> or <Ctrl>C.

UNIX Note In UNIX, you can stop the hexlist with the key or key combination that you normally use to cancel jobs.

Redirection

The screen output can also be redirected to a file. For example,

```
hexlist  mydata  >  hexout  <Enter>
```

will do a hexlist of the file mydata, displaying 75 characters per line and saving the output in the file called hexout.

PSP — PostScript Print Program

Location

Installed in the TPL system directory.

File Name

```
psp.exe  (if Windows)
psp  (if UNIX)
```

Purpose

PSP is a powerful utility for printing regular ASCII character files on a PostScript printer.

Instructions

The program is self-documenting. On the command line, type

```
psp  <Enter>
```

Instructions will be displayed on the screen. Wild cards can be used to print multiple files that have a portion of the name in common. For example, to print all files that have the suffix **.txt**, type:

```
psp  *.txt  <Enter>
```

Note For a line of text that ends with a return character and is longer that the width of the page, PSP will "wrap" the long line, then go to a new line for the following text. For example:

This is the first line of text. It is too long for the page width so it wraps when it is printed.

This is the second line of text. It starts on a new line.

If you want to print this type of text file with PSP, you can get a better result by using the FOR_WORD program to remove the return characters within paragraphs. When you use FOR_WORD, write the output to a temporary file. Then print the temporary file with PSP. For example,

```
for_word  myfile  tempfile  <Enter>
psp  tempfile  <Enter>
```

TO_SHOW (Windows only)

Location

Windows: Installed in the TPL system directory.

UNIX: Not available

File Name

TO_SHOW.EXE

Purpose

When TPL REPORT formats a report in line printer (non-PostScript) mode, it sometimes formats horizontal rules as extensions of other lines of the report. This format will gives the best possible result on any type of line printer. The report can be conveniently reviewed on the screen with the TED, the TPL Editor, because TED is custom-programmed to work correctly with the report format. If, instead, you try to edit the report or display it with other software, the horizontal rules may not display the way you want.

You can use the program TO_SHOW to convert a report file to a format that will work with line printers and editors (word processors), and display correctly on the screen using any display software.

Instructions

On the command line, type

```
TO_SHOW report-in report-out  <Enter>
```

where **report-in** is the original report file and **report-out** is the converted report file.

Example

```
TO_SHOW  EMPLOYEE.REP  EMPLOYEE.SHO  <Enter>
```

APPENDIX K

Character Sets

CHARACTERS AND CODEPAGES

The WIN character sets are recommended for the Windows version; the ISO character sets are recommended for the UNIX version.

The default for Windows is WIN88591. The default for UNIX is ISO88591. To select a different character set, use the CODEPAGE statement described in the Format chapter.

If you want your jobs to give identical results using both the Unix and Windows versions, you should use Windows and ISO codepages with all of the characters you need and use character names rather than character codes in your request.

EURO Symbol

TPL Tables provides full support for the euro symbol provided your printer and computer fonts support it. Windows 2000 may not support the euro symbol but Windows XP and Vista do. Sun Solaris 8 does not support the euro but later versions do.

The **country.tpl** has been changed so the currency symbol is a euro for those countries which have adopted it.

If you look in the codepage files such as win88591.cp, you will see 4 different euro entries, **display_euro**, **pdf_euro**, **eps_euro**, and **psprint_euro**. This is because in certain computer environments the the correct way to specify a euro for one purpose is different from the way to express it for a different purpose. If for example you find that a euro symbol is displayed correctly on the screen but does not convert to a pdf correctly, then you should change the **pdf_euro** code but not the **display_euro** code. If you then use **€** in your request, TPL Tables will select the correct euro code to use for the action you are performing. If you have problems with these, please give us a call.

Mapping of Decimal Values to Postscript Codes for Standard Fonts using CODEPAGE=WIN88591

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		053	5	105	i	157		209	Ñ
002		054	6	106	j	158		210	Ò
003		055	7	107	k	159	Ÿ	211	Ó
004		056	8	108	l	160		212	Ô
005		057	9	109	m	161	¡	213	Õ
006		058	:	110	n	162	¢	214	Ö
007		059	;	111	o	163	£	215	×
008		060	<	112	p	164	¤	216	Ø
009		061	=	113	q	165	¥	217	Ù
010		062	>	114	r	166	¦	218	Ú
011		063	?	115	s	167	§	219	Û
012		064	@	116	t	168	¨	220	Ü
013		065	A	117	u	169	©	221	Ý
014		066	B	118	v	170	ª	222	Þ
015		067	C	119	w	171	«	223	ß
016		068	D	120	x	172	¬	224	à
017		069	E	121	y	173	-	225	á
018		070	F	122	z	174	®	226	â
019		071	G	123	{	175	¯	227	ã
020		072	H	124	l	176	°	228	ä
021		073	I	125	}	177	±	229	å
022		074	J	126	~	178	²	230	æ
023		075	K	127		179	³	231	ç
024		076	L	128	€	180	´	232	è
025		077	M	129		181	µ	233	é
026		078	N	130	,	182	¶	234	ê
027		079	O	131	*ƒ*	183	·	235	ë
028		080	P	132	„	184	¸	236	ì
029		081	Q	133	…	185	¹	237	í
030		082	R	134	†	186	º	238	î
031		083	S	135	‡	187	»	239	ï
032		084	T	136	ˆ	188	¼	240	ð
033	!	085	U	137	‰	189	½	241	ñ
034	"	086	V	138	Š	190	¾	242	ò
035	#	087	W	139	‹	191	¿	243	ó
036	$	088	X	140	Œ	192	À	244	ô
037	%	089	Y	141		193	Á	245	õ
038	&	090	Z	142		194	Â	246	ö
039	’	091	[	143		195	Ã	247	÷
040	(	092	\	144		196	Ä	248	ø
041	)	093	]	145	‘	197	Å	249	ù
042	*	094	^	146	’	198	Æ	250	ú
043	+	095	_	147	“	199	Ç	251	û
044	,	096	‘	148	”	200	È	252	ü
045	-	097	a	149	•	201	É	253	ý
046	.	098	b	150	–	202	Ê	254	þ
047	/	099	c	151	—	203	Ë	255	ÿ
048	0	100	d	152	˜	204	Ì		
049	1	101	e	153		205	Í		
050	2	102	f	154	š	206	Î		
051	3	103	g	155	›	207	Ï		
052	4	104	h	156	œ	208	Đ		

Mapping of Decimal Values to Postscript Codes for Standard Fonts using CODEPAGE=WIN88592

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		053	5	105	i	157	ť	209	Ń
002		054	6	106	j	158	ž	210	Ň
003		055	7	107	k	159	ź	211	Ó
004		056	8	108	l	160		212	Ô
005		057	9	109	m	161	ˇ	213	Ő
006		058	:	110	n	162	˘	214	Ö
007		059	;	111	o	163	Ł	215	×
008		060	<	112	p	164	¤	216	Ř
009		061	=	113	q	165	Ą	217	Ů
010		062	>	114	r	166	¦	218	Ú
011		063	?	115	s	167	§	219	Ű
012		064	@	116	t	168	¨	220	Ü
013		065	A	117	u	169	©	221	Ý
014		066	B	118	v	170	Ş	222	Ţ
015		067	C	119	w	171	«	223	ß
016		068	D	120	x	172	¬	224	ŕ
017		069	E	121	y	173	-	225	á
018		070	F	122	z	174	®	226	â
019		071	G	123	{	175	Ż	227	ă
020		072	H	124	|	176	°	228	ä
021		073	I	125	}	177	±	229	Í
022		074	J	126	~	178	˛	230	ć
023		075	K	127		179	ł	231	ç
024		076	L	128	€	180	´	232	č
025		077	M	129		181	µ	233	é
026		078	N	130	,	182	¶	234	ę
027		079	O	131		183	·	235	ë
028		080	P	132	„	184	¸	236	ě
029		081	Q	133	…	185	ą	237	í
030		082	R	134	†	186	ş	238	î
031		083	S	135	‡	187	»	239	ď
032		084	T	136		188	Ľ	240	đ
033	!	085	U	137	‰	189	˝	241	ń
034	"	086	V	138	Š	190	ľ	242	ň
035	#	087	W	139	‹	191	ż	243	ó
036	$	088	X	140	Ś	192	Ŕ	244	ô
037	%	089	Y	141	Ť	193	Á	245	ő
038	&	090	Z	142	Ž	194	Â	246	ö
039	’	091	[	143	Ź	195	Ă	247	÷
040	(	092	\	144		196	Ä	248	ř
041	)	093	]	145	‘	197	Ĺ	249	ů
042	*	094	^	146	’	198	Ć	250	ú
043	+	095	_	147	“	199	Ç	251	ű
044	,	096	‘	148	”	200	Č	252	ü
045	-	097	a	149	•	201	É	253	ý
046	.	098	b	150	–	202	Ę	254	ţ
047	/	099	c	151	—	203	Ë	255	
048	0	100	d	152		204	Ě		
049	1	101	e	153		205	Í		
050	2	102	f	154	š	206	Î		
051	3	103	g	155	›	207	Ď		
052	4	104	h	156	ś	208	Đ		

Mapping of Decimal Values to Postscript Codes for Standard Fonts using CODEPAGE=WIN88599

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		053	5	105	i	157		209	Ñ
002		054	6	106	j	158		210	Ò
003		055	7	107	k	159	Ÿ	211	Ó
004		056	8	108	l	160		212	Ô
005		057	9	109	m	161	¡	213	Õ
006		058	:	110	n	162	¢	214	Ö
007		059	;	111	o	163	£	215	×
008		060	<	112	p	164	¤	216	Ø
009		061	=	113	q	165	¥	217	Ù
010		062	>	114	r	166	¦	218	Ú
011		063	?	115	s	167	§	219	Û
012		064	@	116	t	168	¨	220	Ü
013		065	A	117	u	169	©	221	İ
014		066	B	118	v	170	ª	222	Ş
015		067	C	119	w	171	«	223	ß
016		068	D	120	x	172	¬	224	à
017		069	E	121	y	173	-	225	á
018		070	F	122	z	174	®	226	â
019		071	G	123	{	175	¯	227	ã
020		072	H	124	|	176	°	228	ä
021		073	I	125	}	177	±	229	å
022		074	J	126	~	178	²	230	æ
023		075	K	127		179	³	231	ç
024		076	L	128	€	180	´	232	è
025		077	M	129		181	µ	233	é
026		078	N	130	‚	182	¶	234	ê
027		079	O	131	*ƒ*	183	·	235	ë
028		080	P	132	„	184	¸	236	ì
029		081	Q	133	…	185	¹	237	í
030		082	R	134	†	186	º	238	î
031		083	S	135	‡	187	»	239	ï
032		084	T	136	ˆ	188	¼	240	ğ
033	!	085	U	137	‰	189	½	241	ñ
034	"	086	V	138	Š	190	¾	242	ò
035	#	087	W	139	‹	191	¿	243	ó
036	$	088	X	140	Œ	192	À	244	ô
037	%	089	Y	141		193	Á	245	õ
038	&	090	Z	142		194	Â	246	ö
039	’	091	[	143		195	Ã	247	÷
040	(	092	\	144		196	Ä	248	ø
041	)	093	]	145	‘	197	Å	249	ù
042	*	094	^	146	’	198	Æ	250	ú
043	+	095	_	147	“	199	Ç	251	û
044	,	096	‘	148	”	200	È	252	ü
045	-	097	a	149	•	201	É	253	ı
046	.	098	b	150	–	202	Ê	254	ş
047	/	099	c	151	—	203	Ë	255	ÿ
048	0	100	d	152	˜	204	Ì		
049	1	101	e	153		205	Í		
050	2	102	f	154	š	206	Î		
051	3	103	g	155	›	207	Ï		
052	4	104	h	156	œ	208	Ğ		

Mapping of Decimal Values to Postscript Codes for Standard Fonts using CODEPAGE=ISO88591

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		053	5	105	i	157	˝	209	Ñ
002		054	6	106	j	158	˛	210	Ò
003		055	7	107	k	159	ˇ	211	Ó
004		056	8	108	l	160		212	Ô
005		057	9	109	m	161	¡	213	Õ
006		058	:	110	n	162	¢	214	Ö
007		059	;	111	o	163	£	215	×
008		060	<	112	p	164	€	216	Ø
009		061	=	113	q	165	¥	217	Ù
010		062	>	114	r	166	¦	218	Ú
011		063	?	115	s	167	§	219	Û
012		064	@	116	t	168	¨	220	Ü
013		065	A	117	u	169	©	221	Ý
014		066	B	118	v	170	ª	222	Þ
015		067	C	119	w	171	«	223	ß
016		068	D	120	x	172	¬	224	à
017		069	E	121	y	173	-	225	á
018		070	F	122	z	174	®	226	â
019		071	G	123	{	175	¯	227	ã
020		072	H	124	|	176	°	228	ä
021		073	I	125	}	177	±	229	å
022		074	J	126	~	178	²	230	æ
023		075	K	127		179	³	231	ç
024		076	L	128	–	180	´	232	è
025		077	M	129	—	181	µ	233	é
026		078	N	130		182	¶	234	ê
027		079	O	131		183	·	235	ë
028		080	P	132		184	¸	236	ì
029		081	Q	133		185	¹	237	í
030		082	R	134		186	º	238	î
031		083	S	135		187	»	239	ï
032		084	T	136		188	¼	240	ð
033	!	085	U	137		189	½	241	ñ
034	"	086	V	138		190	¾	242	ò
035	#	087	W	139		191	¿	243	ó
036	$	088	X	140		192	À	244	ô
037	%	089	Y	141		193	Á	245	õ
038	&	090	Z	142		194	Â	246	ö
039	’	091	[	143		195	Ã	247	÷
040	(	092	\	144	ı	196	Ä	248	ø
041	)	093	]	145	`	197	Å	249	ù
042	*	094	^	146	´	198	Æ	250	ú
043	+	095	_	147	ˆ	199	Ç	251	û
044	,	096	‘	148	˜	200	È	252	ü
045	-	097	a	149		201	É	253	ý
046	.	098	b	150	˘	202	Ê	254	þ
047	/	099	c	151	˙	203	Ë	255	ÿ
048	0	100	d	152		204	Ì		
049	1	101	e	153		205	Í		
050	2	102	f	154	˚	206	Î		
051	3	103	g	155		207	Ï		
052	4	104	h	156		208	Đ		

Mapping of Decimal Values to Postscript Codes for Standard Fonts using CODEPAGE=ISO88592

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		053	5	105	i	157		209	Ń
002		054	6	106	j	158		210	Ň
003		055	7	107	k	159		211	Ó
004		056	8	108	l	160		212	Ô
005		057	9	109	m	161	Ą	213	Ő
006		058	:	110	n	162	¢	214	Ö
007		059	;	111	o	163	Ł	215	×
008		060	<	112	p	164	€	216	Ř
009		061	=	113	q	165	Ľ	217	Ů
010		062	>	114	r	166	Ś	218	Ú
011		063	?	115	s	167	§	219	Ű
012		064	@	116	t	168	¨	220	Ü
013		065	A	117	u	169	Š	221	Ý
014		066	B	118	v	170	Ş	222	Ţ
015		067	C	119	w	171	Ť	223	ß
016		068	D	120	x	172	Ź	224	ŕ
017		069	E	121	y	173	›	225	á
018		070	F	122	z	174	Ž	226	â
019		071	G	123	{	175	Ż	227	ă
020		072	H	124	|	176	°	228	ä
021		073	I	125	}	177	ą	229	Í
022		074	J	126	~	178	†	230	ć
023		075	K	127		179	ł	231	ç
024		076	L	128	–	180	·	232	č
025		077	M	129	—	181	I’	233	é
026		078	N	130		182	ś	234	ę
027		079	O	131		183	•	235	ë
028		080	P	132		184	¸	236	ě
029		081	Q	133		185	š	237	í
030		082	R	134		186	ş	238	î
031		083	S	135		187	ť	239	d’
032		084	T	136		188	ź	240	đ
033	!	085	U	137		189	‰	241	ń
034	"	086	V	138		190	ž	242	ň
035	#	087	W	139		191	ż	243	ó
036	$	088	X	140		192	Ŕ	244	ô
037	%	089	Y	141		193	Á	245	ő
038	&	090	Z	142		194	Â	246	ö
039	’	091	[	143		195	Ă	247	÷
040	(	092	\	144		196	Ä	248	ř
041	)	093	]	145		197	Ĺ	249	ů
042	*	094	^	146		198	Ć	250	ú
043	+	095	_	147		199	Ç	251	ű
044	,	096	‘	148		200	Č	252	ü
045	-	097	a	149		201	É	253	ý
046	.	098	b	150		202	Ę	254	ţ
047	/	099	c	151		203	Ë	255	
048	0	100	d	152		204	Ě		
049	1	101	e	153		205	Í		
050	2	102	f	154		206	Î		
051	3	103	g	155		207	Ď		
052	4	104	h	156		208	Đ		

Mapping of Decimal Values to Postscript Codes for Standard Fonts using CODEPAGE=ISO88599

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		053	5	105	i	157		209	Ñ
002		054	6	106	j	158		210	Ò
003		055	7	107	k	159		211	Ó
004		056	8	108	l	160		212	Ô
005		057	9	109	m	161	¡	213	Õ
006		058	:	110	n	162	¢	214	Ö
007		059	;	111	o	163	£	215	×
008		060	<	112	p	164	€	216	Ø
009		061	=	113	q	165	¥	217	Ù
010		062	>	114	r	166	¦	218	Ú
011		063	?	115	s	167	§	219	Û
012		064	@	116	t	168	¨	220	Ü
013		065	A	117	u	169	©	221	İ
014		066	B	118	v	170	ª	222	Ş
015		067	C	119	w	171	«	223	ß
016		068	D	120	x	172	¬	224	à
017		069	E	121	y	173	-	225	á
018		070	F	122	z	174	®	226	â
019		071	G	123	{	175	¯	227	ã
020		072	H	124	|	176	°	228	ä
021		073	I	125	}	177	±	229	å
022		074	J	126	~	178	²	230	æ
023		075	K	127		179	³	231	ç
024		076	L	128	–	180	´	232	è
025		077	M	129	—	181	µ	233	é
026		078	N	130		182	¶	234	ê
027		079	O	131		183	·	235	ë
028		080	P	132		184	¸	236	ì
029		081	Q	133		185	¹	237	í
030		082	R	134		186	º	238	î
031		083	S	135		187	»	239	ï
032		084	T	136		188	¼	240	ğ
033	!	085	U	137		189	½	241	ñ
034	"	086	V	138		190	¾	242	ò
035	#	087	W	139		191	¿	243	ó
036	$	088	X	140		192	À	244	ô
037	%	089	Y	141		193	Á	245	õ
038	&	090	Z	142		194	Â	246	ö
039	’	091	[	143		195	Ã	247	÷
040	(	092	\	144		196	Ä	248	ø
041	)	093	]	145		197	Å	249	ù
042	*	094	^	146		198	Æ	250	ú
043	+	095	_	147		199	Ç	251	û
044	,	096	‘	148		200	È	252	ü
045	-	097	a	149		201	É	253	ı
046	.	098	b	150		202	Ê	254	ş
047	/	099	c	151		203	Ë	255	ÿ
048	0	100	d	152		204	Ì		
049	1	101	e	153		205	Í		
050	2	102	f	154		206	Î		
051	3	103	g	155		207	Ï		
052	4	104	h	156		208	Ğ		

Mapping of Decimal Values to Postscript Codes for Symbol(Y) Font

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		054	6	107	κ	160	€	213	∏
002		055	7	108	λ	161	ϒ	214	√
003		056	8	109	μ	162	′	215	·
004		057	9	110	ν	163	≤	216	¬
005		058	:	111	ο	164	⁄	217	∧
006		059	;	112	π	165	∞	218	∨
007		060	<	113	θ	166	*f*	219	⇔
008		061	=	114	ρ	167	♣	220	⇐
009		062	>	115	σ	168	♦	221	⇑
010		063	?	116	τ	169	♥	222	⇒
011		064	≅	117	υ	170	♠	223	⇓
012		065	A	118	ϖ	171	↔	224	◊
013		066	B	119	ω	172	←	225	〈
014		067	X	120	ξ	173	↑	226	®
015		068	Δ	121	ψ	174	→	227	©
016		069	E	122	ζ	175	↓	228	™
017		070	Φ	123	{	176	°	229	∑
018		071	Γ	124	|	177	±	230	⎛
019		072	H	125	}	178	″	231	⎜
020		073	I	126	~	179	≥	232	⎝
021		074	ϑ	127		180	×	233	⎡
022		075	K	128		181	∝	234	⎢
023		076	Λ	129		182	∂	235	⎣
024		077	M	130		183	•	236	⎧
025		078	N	131		184	÷	237	⎨
026		079	O	132		185	≠	238	⎩
027		080	Π	133		186	≡	239	⎪
028		081	Θ	134		187	≈	240	
029		082	P	135		188	…	241	〉
030		083	Σ	136		189	⏐	242	⌠
031		084	T	137		190	—	243	⌡
032		085	Y	138		191	↵	244	⎮
033	!	086	ς	139		192	ℵ	245	⌡
034	∀	087	Ω	140		193	ℑ	246	⎞
035	#	088	Ξ	141		194	ℜ	247	⎟
036	∃	089	Ψ	142		195	℘	248	⎠
037	%	090	Z	143		196	⊗	249	⎛
038	&	091	[	144		197	⊕	250	⎜
039	ə	092	∴	145		198	∅	251	⎝
040	(	093	]	146		199	∩	252	⎡
041	)	094	⊥	147		200	∪	253	⎨
042	*	095	_	148		201	⊃	254	⎣
043	+	096		149		202	⊇	255	
044	,	097	α	150		203	⊄		
045	−	098	β	151		204	⊂		
046	.	099	χ	152		205	⊆		
047	/	100	δ	153		206	∈		
048	0	101	ε	154		207	∉		
049	1	102	ϕ	155		208	∠		
050	2	103	γ	156		209	∇		
051	3	104	η	157		210	®		

Mapping of Decimal Values to Postscript Codes for Dingbats(D) Font

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
001		054	✖	107	✳	160		213	→
002		055	✗	108	●	161	❡	214	↔
003		056	✘	109	○	162	❢	215	↕
004		057	✙	110	■	163	❣	216	↘
005		058	✚	111	❑	164	❤	217	→
006		059	✛	112	❒	165	❥	218	↗
007		060	✜	113	❑	166	❦	219	→
008		061	†	114	❒	167	❧	220	➜
009		062	✞	115	▲	168	♣	221	→
010		063	✝	116	▼	169	♦	222	→
011		064	✠	117	◆	170	♥	223	➟
012		065	✡	118	❖	171	♠	224	➟
013		066	☩	119	◗	172	①	225	➡
014		067	✣	120	❙	173	②	226	➢
015		068	✤	121	❚	174	③	227	➢
016		069	✥	122	❯	175	④	228	➤
017		070	✦	123	❮	176	⑤	229	➥
018		071	✧	124	❯	177	⑥	230	➦
019		072	★	125	❝	178	⑦	231	◗
020		073	☆	126	❞	179	⑧	232	➡
021		074	✪	127		180	⑨	233	⇨
022		075	✫	128	€	181	⑩	234	⇨
023		076	✬	129		182	❶	235	⇨
024		077	✭	130		183	❷	236	⇦
025		078	✮	131		184	❸	237	⇨
026		079	✯	132		185	❹	238	⇨
027		080	✰	133		186	❺	239	⇨
028		081	✱	134		187	❻	240	
029		082	✲	135		188	❼	241	⇨
030		083	✳	136		189	❽	242	⇨
031		084	✴	137		190	❾	243	➙
032		085	✵	138		191	❿	244	➘
033	✁	086	✶	139		192	①	245	➛
034	✂	087	✷	140		193	②	246	➶
035	✃	088	✸	141		194	③	247	➷
036	✄	089	✹	142		195	④	248	➛
037	☎	090	✺	143		196	⑤	249	➶
038	✆	091	✻	144		197	⑥	250	➙
039	✇	092	✼	145		198	⑦	251	➛
040	✈	093	✽	146		199	⑧	252	➛
041	✉	094	✾	147		200	⑨	253	➠
042	☛	095	✿	148		201	⑩	254	➾
043	☞	096	❀	149		202	❶	255	
044	✌	097	❁	150		203	❷		
045	✍	098	✪	151		204	❸		
046	✎	099	✱	152		205	❹		
047	✏	100	❄	153		206	❺		
048	✐	101	✳	154		207	❻		
049	✑	102	❆	155		208	❼		
050	✒	103	✳	156		209	❽		
051	✓	104	✼	157		210	❾		
052	✔	105	❇	158		211	❿		
053	×	106	✳	159		212	➔		

Special Character Names for WIN88591

Name	Symbol	Name	Symbol	Name	Symbol
Aacute	Á	endash	–	ordmasculine	º
aacute	á	Eth	Ð	oslash	ø
acircumflex	â	eth	ð	Oslash	Ø
Acircumflex	Â	euro	€	otilde	õ
acute	´	exclamdown	¡	Otilde	Õ
Adieresis	Ä	florin	ƒ	paragraph	¶
adieresis	ä	germandbls	ß	periodcentered	·
ae	æ	guillemotleft	«	perthousand	‰
AE	Æ	guillemotright	»	plusminus	±
Agrave	À	guilsinglleft	‹	questiondown	¿
agrave	à	guilsinglright	›	quotedblbase	„
Aring	Å	hyphen	-	quotedblleft	“
aring	å	Iacute	Í	quotedblright	”
atilde	ã	iacute	í	quoteleft	‘
Atilde	Ã	icircumflex	î	quoteright	’
brokenbar	¦	Icircumflex	Î	quotesinglbase	,
bullet	•	Idieresis	Ï	registered	®
caron	ˇ	idieresis	ï	Scaron	Š
ccedilla	ç	igrave	ì	scaron	š
Ccedilla	Ç	Igrave	Ì	section	§
cedilla	¸	logicalnot	¬	sterling	£
cent	¢	macron	¯	Thorn	Þ
circumflex	ˆ	mu	µ	thorn	þ
copyright	©	multiply	×	threequarters	¾
currency	¤	ntilde	ñ	threesuperior	³
dagger	†	Ntilde	Ñ	tilde	˜
daggerdbl	‡	Oacute	Ó	twosuperior	²
degree	°	oacute	ó	uacute	ú
dieresis	¨	ocircumflex	ô	Uacute	Ú
divide	÷	Ocircumflex	Ô	ucircumflex	û
Eacute	É	odieresis	ö	Ucircumflex	Û
eacute	é	Odieresis	Ö	Udieresis	Ü
Ecircumflex	Ê	oe	œ	udieresis	ü
ecircumflex	ê	OE	Œ	Ugrave	Ù
Edieresis	Ë	Ograve	Ò	ugrave	ù
edieresis	ë	ograve	ò	Yacute	Ý
egrave	è	onehalf	½	yacute	ý
Egrave	È	onequarter	¼	Ydieresis	Ÿ
ellipsis	…	onesuperior	¹	ydieresis	ÿ
emdash	—	ordfeminine	ª	yen	¥

Special Character Names for WIN88592

Name	Symbol	Name	Symbol	Name	Symbol
Aacute	Á	ellipsis	…	perthousand	‰
aacute	á	emdash	—	plusminus	±
Abreve	Ă	endash	–	quotedblbase	„
abreve	ă	eogonek	ę	quotedbllleft	“
Acircumflex	Â	Eogonek	Ę	quotedblright	”
acircumflex	â	Eth	Đ	quoteleft	‘
acute	´	eth	đ	quoteright	’
acute	´	euro	€	quoteright	’
Adieresis	Ä	germandbls	ß	quotesinglbase	,
adieresis	ä	guilsinglleft	‹	racute	ŕ
aogonek	ą	guilsinglright	›	Racute	Ŕ
Aogonek	Ą	hungarumlaut	˝	Rcaron	Ř
breve	˘	hungarumlaut	˝	rcaron	ř
breve	˘	hyphen	-	registered	®
brokenbar	¦	iacute	í	ring	˚
bullet	•	Iacute	Í	Sacute	Ś
cacute	ć	Icircumflex	Î	sacute	ś
Cacute	Ć	icircumflex	î	scaron	š
caron	ˇ	Lacute	Ĺ	Scaron	Š
caron	ˇ	lacute	ĺ	scedilla	ş
Ccaron	Č	logicalnot	¬	Scedilla	Ş
ccaron	č	lquoteright	ľ	Tcaron	Ť
ccedilla	ç	Lquoteright	Ľ	tcedilla	ţ
Ccedilla	Ç	lslash	ł	Tcedilla	Ţ
cedilla	¸	Lslash	Ł	tquoteright	ť
cedilla	¸	mu	µ	uacute	ú
copyright	©	multiply	×	Uacute	Ú
currency	¤	Nacute	Ń	Udieresis	Ü
dagger	†	nacute	ń	udieresis	ü
daggerdbl	‡	ncaron	ň	Uhungarumlaut	Ű
Dcaron	Ď	Ncaron	Ň	uhungarumlaut	ű
degree	°	Oacute	Ó	Uring	Ů
dieresis	¨	oacute	ó	uring	ů
divide	÷	ocircumflex	ô	yacute	ý
dotaccent	˙	Ocircumflex	Ô	Yacute	Ý
dquoteright	ď	Odieresis	Ö	Zacute	Ź
eacute	é	odieresis	ö	zacute	ź
Eacute	É	ogonek	˛	Zcaron	Ž
Ecaron	Ě	ogonek	˛	zcaron	ž
ecaron	ě	Ohungarumlaut	Ő	zdotaccent	ż
edieresis	ë	ohungarumlaut	ő	Zdotaccent	Ż
Edieresis	Ë	periodcentered	·		

Special Character Names for WIN88599

Name	Symbol	Name	Symbol	Name	Symbol
Aacute	Á	Egrave	È	onequarter	¼
aacute	á	ellipsis	…	onesuperior	¹
acircumflex	â	emdash	—	ordfeminine	ª
Acircumflex	Â	endash	–	ordmasculine	º
acute	´	euro	€	Oslash	Ø
adieresis	ä	exclamdown	¡	oslash	ø
Adieresis	Ä	florin	ƒ	Otilde	Õ
ae	æ	gbreve	ğ	otilde	õ
AE	Æ	Gbreve	Ğ	paragraph	¶
Agrave	À	germandbls	ß	periodcentered	·
agrave	à	guillemotleft	«	perthousand	‰
Aring	Å	guillemotright	»	plusminus	±
aring	å	guilsinglleft	‹	questiondown	¿
Atilde	Ã	guilsinglright	›	quotedblbase	„
atilde	ã	hyphen	-	quotedblleft	“
breve	˘	iacute	í	quotedblright	”
brokenbar	¦	Iacute	Í	quoteleft	‘
bullet	•	Icircumflex	Î	quoteright	’
caron	ˇ	icircumflex	î	quotesinglbase	,
ccedilla	ç	idieresis	ï	registered	®
Ccedilla	Ç	Idieresis	Ï	scaron	š
cedilla	¸	Idotaccent	İ	Scaron	Š
cedilla	¸	igrave	ì	scedilla	ş
cent	¢	Igrave	Ì	Scedilla	Ş
circumflex	ˆ	logicalnot	¬	section	§
copyright	©	macron	¯	sterling	£
currency	¤	mu	µ	threequarters	¾
dagger	†	multiply	×	threesuperior	³
daggerdbl	‡	ntilde	ñ	tilde	˜
degree	°	Ntilde	Ñ	twosuperior	²
dieresis	¨	Oacute	Ó	Uacute	Ú
divide	÷	oacute	ó	uacute	ú
dotaccent	˙	Ocircumflex	Ô	ucircumflex	û
dotlessi	ı	ocircumflex	ô	Ucircumflex	Û
Eacute	É	odieresis	ö	Udieresis	Ü
eacute	é	Odieresis	Ö	udieresis	ü
ecircumflex	ê	oe	œ	ugrave	ù
Ecircumflex	Ê	OE	Œ	Ugrave	Ù
Edieresis	Ë	ograve	ò	Ydieresis	Ÿ
edieresis	ë	Ograve	Ò	ydieresis	ÿ
egrave	è	onehalf	½	yen	¥

Special Character Names for ISO88591

Name	Symbol	Name	Symbol	Name	Symbol
Aacute	Á	Egrave	È	ograve	ò
aacute	á	egrave	è	onehalf	½
Acircumflex	Â	emdash	—	onequarter	¼
acircumflex	â	endash	–	onesuperior	¹
acute	´	eth	ð	ordfeminine	ª
acute	´	Eth	Ð	ordmasculine	º
Adieresis	Ä	euro	€	oslash	ø
adieresis	ä	exclamdown	¡	Oslash	Ø
ae	æ	germandbls	ß	Otilde	Õ
AE	Æ	grave	`	otilde	õ
agrave	à	guillemotleft	«	paragraph	¶
Agrave	À	guillemotright	»	periodcentered	·
aring	å	hungarumlaut	˝	plusminus	±
Aring	Å	hyphen	-	questiondown	¿
atilde	ã	iacute	í	registered	®
Atilde	Ã	Iacute	Í	ring	˚
breve	˘	Icircumflex	Î	section	§
brokenbar	¦	icircumflex	î	sterling	£
caron	ˇ	Idieresis	Ï	thorn	þ
Ccedilla	Ç	idieresis	ï	Thorn	Þ
ccedilla	ç	igrave	ì	threequarters	¾
cedilla	¸	Igrave	Ì	threesuperior	³
cent	¢	logicalnot	¬	tilde	˜
circumflex	ˆ	macron	¯	twosuperior	²
copyright	©	mu	µ	Uacute	Ú
degree	°	multiply	×	uacute	ú
dieresis	¨	Ntilde	Ñ	Ucircumflex	Û
divide	÷	ntilde	ñ	ucircumflex	û
dotaccent	˙	Oacute	Ó	Udieresis	Ü
dotlessi	ı	oacute	ó	udieresis	ü
eacute	é	Ocircumflex	Ô	ugrave	ù
Eacute	É	ocircumflex	ô	Ugrave	Ù
ecircumflex	ê	odieresis	ö	Yacute	Ý
Ecircumflex	Ê	Odieresis	Ö	yacute	ý
edieresis	ë	ogonek	˛	ydieresis	ÿ
Edieresis	Ë	Ograve	Ò	yen	¥

Special Character Names for ISO88592

Name	Symbol	Name	Symbol	Name	Symbol
Aacute	Á	emdash	—	quoteright	’
aacute	á	endash	–	Racute	Ŕ
abreve	ă	eogonek	ę	racute	ŕ
Abreve	Ă	Eogonek	Ę	Rcaron	Ř
Acircumflex	Â	Eth	Đ	rcaron	ř
acircumflex	â	eth	ð	ring	˚
acute	´	euro	€	Sacute	Ś
Adieresis	Ä	germandbls	ß	sacute	ś
adieresis	ä	hungarumlaut	˝	Scaron	Š
Aogonek	Ą	Iacute	Í	scaron	š
aogonek	ą	iacute	í	scedilla	ş
breve	˘	icircumflex	î	Scedilla	Ş
Cacute	Ć	Icircumflex	Î	Tcaron	Ť
cacute	ć	Lacute	Ĺ	tcedilla	ţ
caron	ˇ	lacute	ĺ	Tcedilla	Ţ
Ccaron	Č	lquoteright	l’	tquoteright	t’
ccaron	č	Lquoteright	Ľ	uacute	ú
ccedilla	ç	lslash	ł	Uacute	Ú
Ccedilla	Ç	Lslash	Ł	udieresis	ü
cedilla	¸	multiply	×	Udieresis	Ü
Dcaron	Ď	Nacute	Ń	uhungarumlaut	ű
degree	°	nacute	ń	Uhungarumlaut	Ű
dieresis	¨	Ncaron	Ň	uring	ů
divide	÷	ncaron	ň	Uring	Ů
dotaccent	˙	Oacute	Ó	yacute	ý
dquoteright	d’	oacute	ó	Yacute	Ý
eacute	é	Ocircumflex	Ô	Zacute	Ź
Eacute	É	ocircumflex	ô	zacute	ź
Ecaron	Ě	odieresis	ö	Zcaron	Ž
ecaron	ě	Odieresis	Ö	zcaron	ž
edieresis	ë	ogonek	˛	zdotaccent	ż
Edieresis	Ë	ohungarumlaut	ő	Zdotaccent	Ż
		Ohungarumlaut	Ő		

Special Character Names for ISO88599

Name	Symbol	Name	Symbol	Name	Symbol
Aacute	Á	Edieresis	Ë	ograve	ò
aacute	á	egrave	è	Ograve	Ò
Acircumflex	Â	Egrave	È	onehalf	½
acircumflex	â	emdash	—	onequarter	¼
acute	´	endash	–	onesuperior	¹
acute	´	euro	€	ordfeminine	ª
adieresis	ä	exclamdown	¡	ordmasculine	º
Adieresis	Ä	Gbreve	Ğ	Oslash	Ø
AE	Æ	gbreve	ğ	oslash	ø
ae	æ	germandbls	ß	otilde	õ
agrave	à	guillemotleft	«	Otilde	Õ
Agrave	À	guillemotright	»	paragraph	¶
aring	å	hungarumlaut	˝	periodcentered	·
Aring	Å	hyphen	-	plusminus	±
Atilde	Ã	iacute	í	questiondown	¿
atilde	ã	Iacute	Í	quoteright	’
breve	˘	icircumflex	î	registered	®
brokenbar	¦	Icircumflex	Î	ring	˚
caron	ˇ	idieresis	ï	scedilla	ş
ccedilla	ç	Idieresis	Ï	Scedilla	Ş
Ccedilla	Ç	Idotaccent	İ	section	§
cedilla	¸	igrave	ì	sterling	£
cedilla	¸	Igrave	Ì	threequarters	¾
cent	¢	logicalnot	¬	threesuperior	³
copyright	©	macron	¯	twosuperior	²
degree	°	mu	µ	Uacute	Ú
dieresis	¨	multiply	×	uacute	ú
divide	÷	Ntilde	Ñ	Ucircumflex	Û
dotaccent	˙	ntilde	ñ	ucircumflex	û
dotlessi	ı	oacute	ó	Udieresis	Ü
eacute	é	Oacute	Ó	udieresis	ü
Eacute	É	ocircumflex	ô	ugrave	ù
Ecircumflex	Ê	Ocircumflex	Ô	Ugrave	Ù
ecircumflex	ê	odieresis	ö	ydieresis	ÿ
edieresis	ë	Odieresis	Ö	yen	¥
		ogonek	˛		

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z